CONVEX Multibus Storage Module Drive Disk Formatter (*dev4110*) Diagnostics Manual

Document No. 760-002030-000

First Edition
May 1991

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX Multibus Storage Module Drive (SMD)
Disk Formatter (dev4110) Diagnostics Manual
Order No. DHW-232
First Edition

Printed in the United States of America

Revision Sheet

CONVEX Multibus Storage Module Drive (SMD) Disk Formatter (dev4110) Diagnostics Manual

Edition	Document No.	Date	Description
First	760-002030-000	May 1991	First release. Contains the *dev4110* diagnostic test information from the *CONVEX PBUS I/O Systems Diagnostics Manual.*

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

Appendixes

A Reporting Problems

List of Tables

List of Figures

Preface

Purpose and Intended Audience

This manual explains how to run the *dev4110* diagnostic, which formats, verifies the format, and interactively tests up to 12 Storage Module Drives (SMDs) at the same time. This document is not a tutorial, but rather a reference for the users of the *dev4110* diagnostics, including field service and manufacturing test personnel, as well as the diagnostics sustaining staff. In addition, CONVEX customers can use this manual to execute the *dev4110* diagnostic.

Scope

This manual applies to all CONVEX computers.

Organization

This document consists of the following:

- **Chapter 1. Diagnostics Environment**—Introduces the theories and concepts that underlie I/O diagnostics on CONVEX machines as well as the basic overview, philosophy, and structure of I/O diagnostics.
- **Chapter 2. EGOS Overview**—Provides a brief overview of the Event Governed Operating System (EGOS) and how it relates to device and peripheral diagnostics testing.
- **Chapter 3. Dshell Overview**—Provides a brief overview of and a general introduction to the *dshell* utility.
- **Chapter 4. Multibus SMD Disk Formatter and Interactive Test (*dev4110*)**—Describes how to operate the diagnostic, including prerequisites, test invocation, hardware initialization sequence, and class descriptions. It also describes the disk parameters file, diagnostic cylinder, and controller and drive error codes and messages.
- **Appendix A. Reporting Problems**—Provides an example of the CONVEX ***contact*** utility for reporting minor software and hardware problems.

Notational Conventions

The notational conventions used in this text are listed below:

- Bit numbering is left to right, `N-1` through `0`. The most significant numerical bit is `N-1`, the least significant `0`. The bit numbering represents the binary weight of a position.
- Bit fields are specified using the following convention: *name*<*x..y*> where the bit field is *name* from bits *x* through *y*.
- Individual bit positions within a register are denoted by specific positions separated by commas. For example, REG<15,4,0> denotes bits 15, 4, and 0 of REG.
- Byte numbering is from left to right
- A *bit* is a single binary value or entity
- A *nibble* is 4 bits
- A *byte* is 8 bits
- A *halfword* is 16 bits
- A *word* is 32 bits
- A *longword* is 64 bits
- *Single precision* is a 32-bit floating point word
- *Double precision* is a 64-bit floating point longword
- An *instruction* is a multihalfword operand
- A bit is *set* when it contains a binary value of `1`.
- A bit is *clear* when it contains a binary value of `0`.
- All memory and I/O addresses are written in hexadecimal notation unless explicitly stated otherwise.
- All register contents are written in hexadecimal notation unless explicitly stated otherwise.
- A *register* is a programmer-visible hardware storage element internal to the processor
- *Physical memory* is the physical storage installed in the processor
- *Virtual memory* is the perceived amount of physical memory as seen by the application programmer
- The symbol *K* is an abbreviation for *kilo* or 1,024
- The symbol *M* is an abbreviation for *mega* or 1,048,576
- The symbol *G* is an abbreviation for *giga* or 1,073,741,824
- A *stack* is a linked-list group of words useful for dynamic allocation and deallocation of memory
- A *return block* is a collection of registers that is pushed or popped from a context stack in response to an instruction or other event
- *Reserved* or *undefined* convey what to expect, if anything, from unused fields in registers, reserved memory, or reserved I/O space. Algorithm implementation based on the use of undefined or reserved fields is not recommended.

Warnings

The following are examples of warnings, cautions, and notes and their typical content as used in CONVEX documents:

WARNING

Warnings highlight procedures or information necessary to avoid injury to personnel. A warning immediately precedes the critical information and includes a description of the hazard.

CAUTION

Cautions highlight procedures or information necessary to avoid damage to equipment, loss of data, or invalid test results. A caution immediately precedes the critical information and includes a description of the possible damage.

NOTE

Notes highlight useful information that is supplemental in nature. A note may immediately precede or follow the information that is being highlighted.

Associated Documents

The following is a partial list of other manuals or books that may provide more detailed information on the topics presented in this manual:

- *CONVEX Processor Diagnostics Manual (C1, C120)*, Order No. DHW-071
- *CONVEX Processor Diagnostics Manual (C200 Series)*, Order No. DHW-081
- *CONVEX Architecture Reference*, Order No. DHW-005
- *CONVEX SPU UNIX Utilities Manual*, Order No. DHW-021
- *CONVEX Processor Operation Guide (C100 Series, C200 Series)*, Order No. DHW-015
- *CONVEX Diagnostic Utilities Manual (C1, C120)*, Order No. DHW-072
- *CONVEX Diagnostic Utilities Manual (C200 Series)*, Order No. DHW-082
- *CONVEX UNIX Tutorial Papers*, Order No. DSW-002
- *The C Programming Language*, Kernighan & Ritchie, Order No. DSW-046

Ordering Documentation

To order the most current version of this or any other CONVEX document, use the product number. If the product number is not known, order by the exact title. In some situations, the most current version may not be desired. To receive a specific version of a manual, order the manual by its document number, or part number, which can be obtained by contacting the local CONVEX office or by calling the Technical Assistance Center.

The product number for this manual is DHW-232.
The document number for this manual is 760-002030-000.

CONVEX documents can be ordered by mail by sending a request to:

CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson TX 75083-3851 USA

Technical Assistance

Hardware and software support can be obtained through the CONVEX Technical Assistance Center (TAC):

- From all locations in the continental United States, call 1(800)952-0379.
- From locations in Alaska, Hawaii, and Canada, call 1(214)497-4379.
- From all other locations, contact the nearest CONVEX office.

Reader's Forum

If you wish to mail your comments to us, please use the form at the end of this manual and list the document page number with your questions and comments. Thank you.

Chapter 1
Diagnostics Environment

1.1 Overview

CONVEX system diagnostics consist of a suite of test programs designed (except where noted) to execute under the Service Processor operating system, SPU UNIX. These programs utilize the capabilities of the Service Processor to test the operation of one or more of the functions of the system and report any errors detected. All of the diagnostics in this manual are intended to be executed "off-line"; that is, while CONVEX UNIX is not being executed by any of the Central Processing Units (CPUs) in the system.

The Service Processor, together with SPU UNIX, various diagnostic utilities, and the test programs, themselves, comprise the CONVEX diagnostic environment. This chapter describes the hardware and software components of this environment and is intended to provide the background necessary to fully utilize the capabilities of the CONVEX processor diagnostics.

For more information about the diagnostic environment refer to the Diagnostic Environment chapter in the *CONVEX Processor Diagnostics Manual (C200 Series)* or the *CONVEX Processor Diagnostics Manual (C1, C120)* depending on the architecture of the machine under test.

1.2 Test Program Naming Conventions

Test program names are in the form *cattypedevnn.suffix* where:

- *cat* is the subsystem being tested
- *type* is the type of test being performed, e.g., standalone, self-test, or offline functional test
- *dev* is the device being tested, e.g., disk, tape, or printer. This segment of the test program name is used *only* if the category is a device.
- *nn* is a CONVEX code used for distinguishing between test programs
- *suffix* is one of three program identifiers:
  - *.t* are programs that execute on SP2
  - *.x00* and *.rnn* are object files for different target processors other than the SP2. The target processor depends on the subject of the test. The test program name must have the test program category (*cat*) at the beginning of the name to determine the target processor.

1.2.1 Test Program Categories

Test program categories include those tests for the CPU, peripheral devices, I/O system, memory system, SP2, and entire system. For example, *cpu4041* is a CPU vector instruction test while *mem4000* is a memory system functional test. The following table lists test program categories:

Table 1–1, Test Program Categories

TEST PROGRAM CATEGORIES	
Test Category (*cat*)	**Description**
cpu	CPU subsystem related test
dev	Peripheral device test
io, idc, tli	I/O subsystem related test
mem	Memory subsystem related test
spu	SP2 subsystem related test

1.2.2 Test Program Types

A test program type describes whether a test is a standalone test, self-test, kernel hardware test, or an offline or online functional test. See the following table for the numbering system and description of test program types:

Table 1–2, Test Program Types

TEST PROGRAM TYPES	
Number (*type*)	**Description**
0	Standalone test
1	Self-test
2	Kernel hardware test
4, 5	Offline functional test

1.2.3 Test Program Device Types

Test programs will test disks, tapes, terminals, printers, and networks. See the following table for the numbering scheme and a description of the test program device types:

Table 1–3, Test Program Device Types

TEST PROGRAM DEVICE TYPES	
Number (*dev*)	**Description**
1	Disk
2	Tape
3	Terminal
4	Printer
5	Network

1.2.4 Examples of Test Program Names

The following table presents some examples using the naming conventions outlined above:

> **NOTE**
>
> In the following table, SOFF stands for Standard Object File Format.

Table 1–4, Example Test Program Names

EXAMPLE TEST PROGRAM NAMES	
Test Program Name	**Description**
cpu4041.t	SP2 object code in *b.out* format for *cpu4041*
cpu4041.rnn	C210 or C220 machine object code in SOFF format (relocatable)
cpu4041.x00	C210 or C220 machine object code in SOFF format (linked to run in segment 0)
mem4000.t	SP2 object code in *b.out* format for *mem4000*
mem4000.x00	C210 or C220 machine object code in SOFF format (linked to run in segment 0)
dev4100.t	SP2 object code in *b.out* format for *dev4100*
dev4100.x00	IOP object code in *b.out* format

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2
EGOS Overview

2.1 Overview

This chapter provides an overview of the Event Governed Operating System (EGOS) and how it relates to device and peripheral diagnostics testing. There are three basic types of EGOS systems, one for each type of CCU. There is one for the Multibus interface, one for the VME interface, and one for the HIA interface. This chapter will explain the three types of EGOS systems and how EGOS is positioned within the overall operating system environment.

2.2 Purpose of EGOS for Diagnostic Testing

EGOS is basically a simple operating system that the device tests use to handle interrupts, schedule processes, and generally allocate and control IOP/VIOP resources. The diagnostics code uses both EGOS and the Message Based System (MBS) to manipulate test program control over to the CCU side of the test program. MBS is not a part of EGOS but rather a system that allows a common section of memory to be used as a message area between multiple processors. For more information on MBS, refer to the *CONVEX Guide to Writing Device Drivers.*

EGOS initially sets up interrupt tables, determines how many chassis there are, and initializes its windows and resource allocation tables.

2.3 EGOS for the Multibus Interface

EGOS for the Multibus interface supports event driven device drivers. The Multibus version of EGOS takes interrupts that are local to a CCU and channels those errors to the proper piece of code to handle the error. It basically supplies the error interrupt handlers for the CCU error interrupts. It also contains support routines to control allocation of the various CCU-related resources.

2.4 EGOS for HSP Interface, HSP EGOS

EGOS for the HSP interface supports event driven device drivers. The HSP version of EGOS is like the Multibus version. It takes interrupts that are local to a CCU and channels those errors to the proper piece of code to handle the error. It basically supplies the error interrupt handlers for the CCU error interrupts. It also contains support routines to control allocation of the various CCU-related resources.

2.5 EGOS for VME Interface, VIOP EGOS

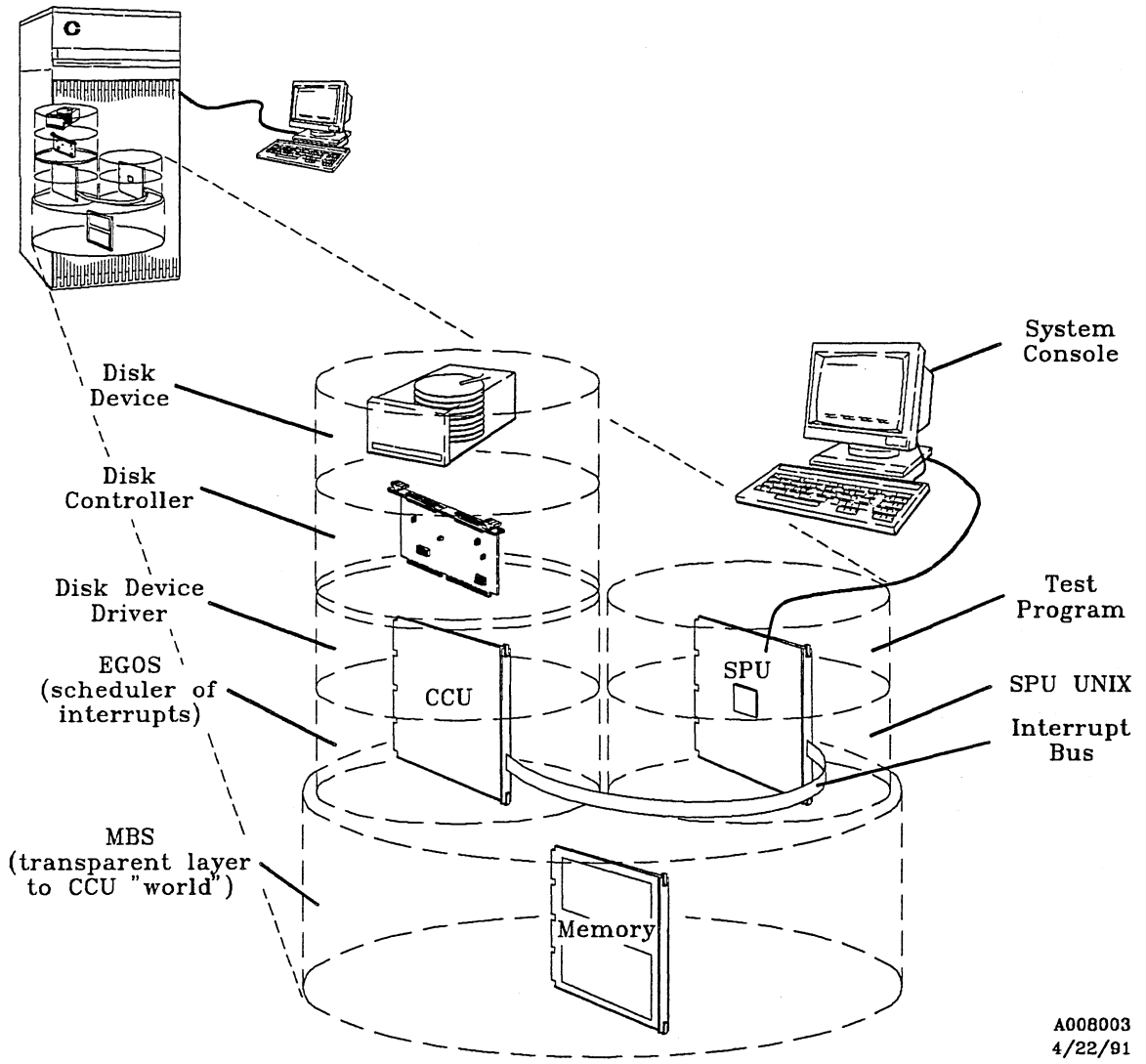
The VME interface version of EGOS is designed with a scheduler for the VIOP and is called VIOP EGOS. VIOP EGOS supports event driven device drivers as well as process type device drivers. VIOP EGOS utilizes a *sleep/wakeup* type of process control that improves efficiency of the device driver and makes it less complicated to create user written device drivers. Each process device driver has a priority level that can be defined relative to other processes. The scheduler supports 32 process priorities and is preemptive for higher priority processes. The VIOP hardware supports 14 device events for event driven device drivers. The 14 levels actually share 2 68020 interrupt levels. Therefore, two is the maximum number of processes at any given time.

2.6 EGOS Position in the Environment

EGOS is positioned in the operating environment between the actual device driver and MBS. MBS is a transparent layer that bridges the CCU and its resources to SPU UNIX. SPU UNIX handles many of the message manipulations that occur during testing. Many error messages that occur during diagnostics testing come from the device driver. When the device driver detects an error from the controller, it calls a routine in EGOS that places a message in the MBS system. This causes SPU UNIX to be interrupted and it retrieves the message from MBS. SPU UNIX then passes a signal to the test program. The test program then prints an error message to the console based on the code that it received.

The following figure illustrates the position of EGOS in the operating system environment.

Figure 2–1, EGOS' Position in the Environment



THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3
Dshell Overview

3.1 Overview

This chapter provides a brief overview of the *dshell* utility. Included in this overview is an overall explanation of the utility and a list of the utility's commands. For a complete description of this utility, refer to the Dshell chapter of the *CONVEX Diagnostic Utilities Manual (C200 Series)* or the *CONVEX Diagnostic Utilities Manual (C1, C120)* depending on the architecture of the machine under test.

3.2 Diagnostic Shell (*dshell*) Overview

The Diagnostic Shell (*dshell*) is a command interface program that runs on the Service Processor. Most of the diagnostics available for the CONVEX machines are interfaced through the *dshell*. Certain peripheral diagnostics are run as standalone tests. To determine whether a test can be run under the *dshell*, consult the appropriate chapter in this manual.

The *dshell* has two basic functions:

- Selecting diagnostics for execution
- Selecting test options
  - Pause on a failure or at the beginning or end of any specific subtest
  - Loop on a specific type of subtest or on a given set of subtests
  - Select subtest execution order
  - Direct test output to a file or to the screen (or both) to monitor the test as it runs or to analyze test results later
  - Select long or short error messages, or turn messages off
  - Execute either user-created or predefined command scripts

The following table list the various *dshell* commands and their functions.

Table 3–1, *dshell* Commands

COMMAND	FUNCTION
! [command]	This command is used to access, or *fork* a UNIX shell to execute the command that follows *!*.
exit	The *exit* command causes immediate termination of the *dshell* process and any test processes that may have been forked.
quit	The *quit* command causes immediate termination of the *dshell* process and any test processes that may have been forked.
^C	Returns user to the *dshell* command level if no subtest is running.
^B	Immediately terminate the *dshell* and any associated active processes. Core is dumped.
help	The *help* command causes a standard *help* menu to be displayed. The menu describes the correct command syntax for each *dshell* command and gives a terse description of what each command does.
status	The *status* command generates a report on the current state of the *dshell* command options. This report gives the name of each flag, its current value, and an explanation of its current effect.
log [options]	The *log* command provides a mechanism for specifying the number of failures that will be allowed to occur before a test or subtest terminates execution.
loop [options]	The *loop* command causes the *dshell* to repeat the execution of a test or subtest.
msgs [options]	The *msgs* command enables or disables different levels of test, class, and subtest result messages.
pause [options]	The *pause* command returns program control to the *dshell* to the beginning, end, or failure of all or specific subtests.
test [options]	The *test* executes specific tests, and displays test, class, and subtest menus.

3.3 Syntax Help for *dshell* Commands

The syntax for each *dshell* command can be obtained by typing the command with no options and pressing <**CR**>. For example, by entering **loop** and pressing <**CR**>, the syntax help in the following figure will be displayed on the screen:

Figure 3–1, Syntax Help for the *loop* Command

```
: loop
Proper syntax is:

loop off (-s) (-t)                          :disables loop modes
loop -s nnn                                 :loop on subtest nnn
loop -t                                     :loop on test
```

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4
Multibus Storage Module Drive (SMD) Disk Formatter and Interactive Test (*dev4110*)

4.1 Overview

The *dev4110* test is capable of formatting, verifying the format, and interactively testing up to 12 Storage Module Drives (SMDs) at the same time. It also allows manual input of a Manufacturer's Defect (MD) map before or after a format is performed.

4.2 Prerequisites and Required Equipment

The *dev4110* test is dependent on the existence and correctness of the SPU disk file */mnt/bin/lib/DB_diskfmt*. Refer to the section, "Disk Parameters File, *DB_diskfmt* Description" in this test description for more information about this file.

The test requires the system configuration in one of the next two tables depending on the type of machine under test.

Table 4–1, Hardware Requirements (C1, C120)

ITEM	MINIMUM	MAXIMUM
Xylogics 450/451 controller	1	12
SMD device	1	12
Input/output processor (IOP)	1	5
Multibus card cage	1	10 (2/IOP)
Service processor unit (SPU)	1	1
Multibus control unit (MBCU)	1	12
Memory control unit (MCU)	1	1
Memory array unit (MAU)	4MB	System limit

Table 4–2, Hardware Requirements (C200 Series)

ITEM	MINIMUM	MAXIMUM
Xylogics 450/451 controller	1	12
SMD device	1	12
Input/output processor (IOP)	1	12
Multibus card cage	1	12
Service processor unit (SP2)	1	1
Multibus control unit (MBCU)	1	12
Memory system (one odd, one even)	1	System Limit
Peripheral interface adapter (PIA)	1	4
CPU utilities (CPX)	1	1

4.3 Test Invocation

The *dev4110* test executes under the Diagnostic Shell (*dshell*) and supports all the features of the *dshell*. The *dshell* permits tests to be initiated in any order.

To invoke the *dev4110* test, use the procedure shown in the following figure. All responses in **boldface** are entered by the user. The prompts and responses appear sequentially on the screen, one line at a time. All prompts and responses are shown in one figure for convenience.

Figure 4–1, *dev4110* Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test dev4110 [-c [class number(s)]] [-s [subtest number(s)]] [+>filename] RETURN
```

NOTE

After entering **dshell**, specific *dshell* parameters may be changed. Refer to the "Dshell Overview" chapter of this manual for more information.

Entering only **test dev4110** executes all *dev4110* subtests sequentially except for Class 2 (same as Subtest 200). Execute a specific class(es) of subtest(s) or one or more individual subtests by using the **-c** or **-s** options, respectively. Detailed information for using these options can be found in the "Dshell Overview" chapter of this manual. The [+>*filename*] option allows the test results to be appended to *filename*.

NOTE

The Class 2 (Subtest 200) Interactive Test is only started by specifying the **-c** or **-s** options. Only the Class 1 subtest runs if only **test dev4110** is entered.

The following alternate test invocation procedure may be required in some cases.

CAUTION

The user response, **initall**, is typically required if the *initall* utility has not been run since the last powerup. However, if any problems have occurred subsequent to the last time *initall* was run, (i.e., system crash or hard error), it should be run again. In this case, failure to run *initall* could result in invalid test results. The *initall* utility requires a significant amount of time (2 to 3 minutes depending on if the control stores have been previously loaded) to execute. The control stores do not need to be loaded for disk tests. If no system abnormalities have occurred subsequent to the last time the system was booted or *initall* was executed, it is not necessary to run *initall.*

Figure 4–2, Alternate Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test dev4110 [-c [class number(s)]] [-s [subtest number(s)]] [+>filename] RETURN
```

4.3.1 Test Parameter Menu

Once the test is invoked, a test menu prompt is presented allowing selection of default switches. The following figure shows all prompts, their possible answers (in brackets [ ]), and their default answers (in parentheses ( )). The prompts and responses in the following figure appear sequentially on the screen, one line at a time. All the prompts and responses are shown in one figure for convenience.

For help or information during test parameter entry, enter one of the following characters followed by a RETURN:

Table 4–3, Getting Help During Test Parameter Entry

Character	Description
?	Displays this help menu
d	Displays *DB_diskfmt* file containing the drive parameters
e	Displays entered drive entries
h	Provides help for a specific prompt
i	Displays the */ioconfig* file

After the desired help information displays, the system beeps and redisplays the last prompt.

The **Test Parameter Menu** illustrates *all* questions that can be displayed during test parameter input. However, some questions may be omitted, depending on answers to previous questions. In all cases, questions are numbered sequentially. However, the numbers displayed on the screen during testing may not correspond to those shown in the example **Test Parameter Menu**, as the questions illustrated are examples only.

Figure 4–3, Test Parameter Menu

```
                    ENTER TEST PARAMETERS
       []     Encloses allowed input ranges or values
       ()     Encloses the default value
       ^      Returns to the previous prompt
       :nn    Returns to the prompt # nn
       :      Returns to the first unsatisfied prompt
       :?     Reviews previous entries
?      Prints an additional help menu
 1:  Number of errors allowed per device each subtest
    [0-65535]                                        (0) -> RETURN
 2:  Number of devices failed after which test aborts
    [0-65535]                                       (12) -> RETURN
 3:  Verbosity of test [0-65535]                     (3) -> RETURN
 4:  Ioconfig file []                      (/ioconfig) -> RETURN
                PERIPHERAL CONFIGURATION DATA[1]
     CCU    Chassis  Type    CSR   Int Unit Type
    -------- ------- -------- ----- --- ---- --------
 1) iop  4      0    DKC-001  0x3f0  2   0   DKD-008
 2) iop  4      0    DKC-001  0x3f0  2   1   DKD-005
 3) iop  4      0    DKC-001  0x3f8  2   1   DKD-005
 4) iop  4      0    DKC-001  0x3f8  2   0   DKD-008
 5) iop  4      0    DKC-001  0x3f8  2   1   DKD-008
Enter device 99 to begin user-defined configurations or 0 to end selection
 5:  Device selection [0,1-5,99]                     (0) -> 1 RETURN
 6:  Device selection [0,1-5,99]                     (0) -> 99 RETURN
Enter IOP -1 to end user-defined configurations
 7:  IOP [0,3-7]                                     (0) -> 3 RETURN
 8:  Multibus Chassis [0-3]                          (0) -> RETURN
 9:  Controller Offset in Multibus [0x0-0xfff]
                                                (0x3f0) -> 3f8 RETURN
10:  Interrupt number [0-7]                          (2) -> 3 RETURN
11:  Unit number [0-3]                               (0) -> RETURN
12:  Drive name []                            (DKD-005) ->[2] RETURN
Enter IOP -1 to end user-defined configurations
13:  IOP [0,3-7]                                     (0) -> 1 RETURN
14:  Enter OK, or :NN to return to question NN [OK]
                                                    (OK) -> RETURN

          **********  WARNING!  **********
       THIS TEST IS DATA DESTRUCTIVE!
       IF YOU CONTINUE, DATA WILL BE LOST!

************************************************************
* SET WRITE PROTECT ON ALL DRIVES YOU DON'T WANT FORMATTED *
************************************************************
Do you want to continue [yn]  -> y RETURN
```

[1] This information is only displayed once; to display it again, type **i** RETURN at any prompt during test parameter query.

[2] The drive name must match with a drive in the disk parameters file */mnt/bin/lib/DB_diskfmt*). To display the disk parameters file, type **d** RETURN at any prompt.

At any time during the test parameter sequence, several options are available as denoted at the top of the Test Parameter Menu. The following list summarizes the available options:

:nn — Returns to an earlier prompt (n is the prompt number)

: — Advances to the next unanswered prompt

:? — Displays (reviews) all responses up to the current prompt

? — Requests help for the current prompt (if available)

^ — Returns to the previous prompt

4.3.2 Prompt Explanations

The test parameter prompts are repeated and explained in the following paragraphs.

```
Number of errors allowed per device each subtest
[0-65535]                                          (0) ->
```

The number of errors that can occur during a subtest and still allow a device to continue can be specified. For instance, if **3** is entered, a device can have 0-3 errors in any subtest and continue running. On the fourth error, the device stops and is not restarted for the duration of the test. In this example, as long as no more than 3 errors in each subtest occur, the device continues.

> **NOTE**
>
> Subtest 200, Interactive Test reports errors but does not increment the error count when a drive or controller error occurs. In addition, Subtest 100, Input Defects Before Formatting does not affect the error count during pattern test for sector related errors.

```
Number of devices failed after which test aborts
[0-65535]                                         (12) ->
```

Each time a unit fails (exceeds the number of errors allowed each subtest), the count of the number of failed devices is incremented. When a controller fails, the count is incremented for the controller and for each active drive on the controller. If this count exceeds the number of devices specified at this prompt, the entire *dev4110* test aborts. The default number is 12 so the test normally continues even though drives have failed.

> **NOTE**
>
> Drive errors are not counted during the Subtest 200, Interactive Test so failures are not counted during this test.

```
Verbosity of test [0-65535]                    (3) ->
```

The amount of information to be printed can be selected in addition to the standard subtest execution lines. Choose values of the items to print from the table below and add the values. The sum is the number to enter.

Table 4–4, Test Verbosity Levels

VALUE	DESCRIPTION
1	Print summary of all bad sectors found – printed near the end of Subtest 101, Format and Pattern Test.
2	Print each bad sector as it is found in Subtest 101 – may print same sector once for each pattern.
4	Print read of the BBT at startup of Subtest 103, Verify System Format, Subtest 104 Test Diagnostic Cylinder, and Subtest 200, Interactive Test.
8	Print pattern and pass counter in Subtest 200, Interactive Test pattern test. Only the pass counter is normally displayed as an indicator of how far along the test is toward completion.

NOTE

If each bad sector is to be printed as it is found, the following errors are printed.

- 0x05 (header not found)
- 0x06 (Hard ECC error)
- 0x1e (Correctable ECC error)

Any other reported errors than those listed above are countable errors and increment the number of errors per subtest counter. Countable errors are always retried until successful or the number of errors per subtest counter exceeds the number allowed.

The following is an example of verbose output for a value of 1:

Figure 4–4, Verbose Output Screen — Level 1

```
SUMMARY OF BAD BLOCKS FOR ccu/mb/csr/d=3/0/3f0/0

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
         ---------------------- no sectors ----------------------

LOC:  *G=gap  *D=dup  *S=spare  *I=invalid hdr  *B=hdr marked bad

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
         ---------------------- no sectors ----------------------

LOC:  *G=gap  *D=dup  *S=spare  *I=invalid hdr  *B=hdr marked bad

SUMMARY OF BAD BLOCKS FOR ccu/mb/csr/d=3/0/3f0/1

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
 ------------------------------ no sectors --------------------------------

LOC:  *G=gap  *D=dup  *S=spare  *I=invalid hdr  *B=hdr marked bad

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
  22  3   7                  SECC PGM | 372 12   0                  HECC PGM
  51 14  28                  HECC PGM | 523 18   2                  SECC PGM
  77  9  44                  SECC PGM | 544  6  11                   HNF PGM
  78  9  44                   HNF PGM | 591  0   7                  HECC PGM
  79  9  44                  SECC PGM |

LOC:  *G=gap  *D=dup  *S=spare  *I=invalid hdr  *B=hdr marked bad
```

The following is an example of verbose output for a value of 2:

Figure 4–5, Verbose Output Screen — Level 2

```
    subtest 101  0:22:14  Pattern 1: 00000000

dev4110 ERROR (0x1E)-ccu/mb/csr/d=3/0/3F0/1 Correctable ECC error
  Cyl:  22 Hd: 3 Sect:  7
 0:23:52

dev4110 ERROR (0x06)-ccu/mb/csr/d=3/0/3F0/1 Hard ECC error
  Cyl:  51 Hd:14 Sect: 28
 0:29:45 Pattern 2: FFFFFFFF

  ----- More errors will probably be reported during other patterns.
  ----- In fact the errors shown above will probably repeat.
```

The following is an example of verbose output for a value of 4:

Figure 4–6, Verbose Output Screen — Level 4

```
--------------- BAD BLOCK TABLE READS -----------------
Drive: ccu/mb/csr/d - 3/0/3f0/0
        Reading copy #0 starting at cyl:758 hd:18 sec:34    - failed
        Reading copy #1 starting at cyl:758 hd:18 sec:39    - successful
Drive: ccu/mb/csr/d - 3/0/3f0/1
        Reading copy #0 starting at cyl:758 hd:18 sec:34    - successful
```

The following is an example of verbose output for a value of 8:

Figure 4–7, Verbose Output Screen — Level 8

```
(D1;Save)-> p 100 times 0 0 0 to end
pass: 1 pattern: ffffffff
```

With a verbosity setting of 8, the `pass: 1` and the `pattern: ffffffff` line print. Both the pass number and pattern change as the test progresses.

```
Ioconfig file []                          (/ioconfig) ->
```

This prompt allows a file to be chosen that contains peripheral device descriptions. The default displays the system's configuration file. To display an I/O configuration file previously created,

enter the filename. Then the configuration file is displayed. (A relative or absolute path may be specified.)

If drives are to be formatted or tested with a configuration other than the expected configuration, a new I/O configuration file may be created. As an alternative, nonstandard configurations may be input by responding to the user-defined configuration test parameters (questions 7-13 in the "Test Parameter Menu" figure). Then the drive and/or controller does not have to be defined in an I/O configuration file. Drives can be selected from an I/O configuration file and then drives can also be defined with user-defined configurations.

```
Device selection [0,1-5,99]                    (0) ->
```

Enter the number of the drive to be tested. One drive can be selected with each prompt for device selection and up to a total of 12 drives can be selected in any order. Refer to the *Hardware Requirements (C1, C120)* table and the *Hardware Requirements (C200 Series)* table for restrictions on hardware configurations.

> **NOTE**
>
> If drives are formatted for system use, the drives should be on separate controllers. Each controller must format drives sequentially so two on one controller doubles the format time.

If **0** is entered, the selection is terminated and the prompt `Enter OK` is displayed. If **99** is entered, the user-defined prompts are displayed, beginning with the following prompt.

```
IOP [0,3-7]                                    (0) ->
```

This prompt begins the user-defined drive prompts, so if all the drives to be used are in the *ioconfig* file, reply RETURN to end the user-defined drive prompts. However, to enter a unique drive configuration (not in the *ioconfig* file), enter the CCU slot number for the desired IOP. Then additional prompts are displayed requesting hardware configuration information.

```
Multibus Chassis [0-3]                         (0) ->
```

Enter the number of the chassis to be tested. (The drive is attached to a controller which is in a Multibus chassis.)

```
Controller Offset in Multibus [0x0-0xfff]
                                          (0x3f0) ->
```

Enter the low-order 12 bits of the controller's address within the Multibus (this address is selected with switches on the controller).

```
Interrupt number [0-7]                         (2) ->
```

Enter the interrupt level of the controller within the Multibus (this is selected with switches on the controller).

```
Unit number [0-3]                                    (0) ->
```

Enter the unit number of the drive to be tested. The unit number to enter is also the unit selected on the drive (Up to four drives can be attached to one controller; each drive is assigned a unit number, which is usually switch selectable on the drive). For maximum performance, each drive should be on a separate controller although it is not a requirement.

```
Drive name                                    (DKD-005) ->
```

Enter the drive name. The response for this query must be in the form of DKD-xxx and the name must match one of the drive definitions found in the disk parameters file (*/mnt/bin/lib/DB_diskfmt*).

```
IOP [0,3-7]                                          (0) ->
```

Enter the CCU slot number for the desired IOP to set up test parameters for another user-defined drive. Enter a **-1** or RETURN to terminate test parameter entry.

```
Enter OK or :NN to return to question NN [OK]
                                                     (OK) ->
```

To return to a preceding question at anytime during the test parameter sequence, enter CTRL and RETURN. To return to a specific question and make any further changes, a colon and a question number may be entered, for example, **:3**.

If **OK** or RETURN is entered, the test parameter menu terminates and all inputs are no longer changeable.

```
Do you want to continue [yn]          ->
```

Enter RETURN to begin test execution; or enter **n** to abort the *dev4110* test.

When all prompts have been answered, the screen displays a test parameter summary which echos the prompts that have been answered. The following figure illustrates an example of a "Test Parameter Summary" screen. The actual values and responses vary according to the input.

Figure 4–8, Sample Test Parameter Summary

```
              TEST PARAMETER SUMMARY

Number of errors allowed per device each subtest      : 0
Number of devices failed after which test aborts      : 12
Verbosity of test                                     : 3
Ioconfig file                                         : /ioconfig
Enter OK, or :NN to return to question NN             : OK

                     DRIVE CONFIGURATION DATA

      IOP MBus MBus  Int   Unit  #    #  Phys  Log
       #   #   CSR  Level   #   Cyl  Hds  Sec  Sec  Name
      --- --- ----- -----  ---  ---  ---  ---  ---  -------
   1.  4   0  0x3f0   2     0  1024   27   68   67  DKD-008
1000.  4   0  0x3f8   3     0   760   19   60   59  DKD-005

Performing sysreset of ccus and memory...
Initializing I/O subsystem and Loading IOP(S)...

************************************
* SYSTEM FORMAT IS ABOUT TO START    *
* VERIFY WRITE-PROTECT IS SET ON ANY *
* DRIVES YOU ARE NOT FORMATTING      *
************************************

Then reconfirm that you want to continue    [yn] ->
```

4.4 Hardware Initialization Sequence

After the last prompt is entered, and before test code execution, the following events occur:

- A sysreset is performed
- Main memory is allocated for the test
- SPU windows to main memory are initialized
- SPU local test variables are initialized
- The IOP is booted and loaded
- A driver on the IOP is started
- IOP local test variables are initialized

After all the above events have occurred, the test code is started.

4.5 Class Descriptions

The *dev4110* test contains the following two classes of tests.

Table 4–5, *dev4110* Test Classes

CLASS	DESCRIPTION
1	System format and verification tests
2	Interactive test

4.5.1 Class 1 Subtests

Class 1 subtests allow a system format to be written to an SMD and to be verified. The proper way to format a disk is to execute all subtests in the default order.

> **NOTE**
>
> Some individual functions performed during formatting are contained in two separate subtests in order to assist in special testing of drives. Therefore, Subtest 101, Format and Pattern Test should not be invoked unless Subtest 102, Initialize Diagnostic Cylinder is also invoked.

If only Subtest 101, Format and Pattern Test is invoked, the option to quit or continue is available. The test displays the warning message shown in the following figure if only Subtest 101 is invoked:

Figure 4–9, System Format and Verification Warning Screen

```
WARNING! SMD Disk Formatter, dev4110, should never be invoked with
         Subtest 101 selected and no Subtest 102.
Do you want to continue [yn]     - y (RETURN)


                                WARNING!!

        YOU HAVE CHOSEN TO ONLY PARTIALLY FORMAT A DRIVE!
        THE DRIVE WILL NOT BE ACCEPTABLE FOR SYSTEM USE!
```

The Class 1 subtests are shown in the following table:

Table 4–6, Class 1 Subtests

SUBTEST	DESCRIPTION	TIME (hr:min:sec)
100	Input Defects Before Formatting	N/A
101	System Format of SMD	2:20:00[2]
102	Initialize Diagnostic Cylinder	3:00[1]
103	Verify System Format of SMD	5:30[2]
104	Test Diagnostic Cylinder	:15[1]

[1] This test is run sequentially on each drive so multiply this time by the number of drives for an estimate of the required time for this subtest.

[2] Times will double, triple, or quadruple for 2, 3 or 4 drives, respectively, per controller.

4.5.1.1 Subtest 100, Input Defects Before Formatting

Subtest 100 is used to enter defect map information before pattern testing. The inputs can be entered or can be read from a specified file. In either case, the inputs can be in Byte-Count-After-Index (BCAI) format or in logical sector number format.

A command processor is invoked when this subtest begins. Several commands are available for the input of defects. Startup of this subtest is shown in the following figure:

Figure 4–10, Subtest 100 Startup Screen

```
Type 'h' for help with commands
        Logical sector input mode selected.

If you do not have defects to enter, just enter 'q'

make_bad (D1;Sector)->
```

Enter **h** to display a help facility which lists the available commands. The help output is displayed as shown in the following figure:

Figure 4–11, Subtest 100 Help Screen

```
Commands:
c[hange_mode]          - toggles logical sector/defect map entry mode
cyl hd sec             - logical sector to be slipped
cyl hd bcai len        - defect map entry to be slipped
del[ete] cyl hd sec    - delete logical sector entry
del[ete] cyl hd bcai len - delete defect map entry
dr[ive] [n]            - change/display current drive
f[ile] filename        - get inputs from specified file
h[elp]                 - display this information
l[ist]                 - list inputs made so far
q[uit]                 - exit
!UNIX-command          - execute UNIX command
'comment               - comment; ignored
```

When manually entering defects, first select the entry mode. The default mode is logical sector mode. To enter manufacturer's defects (which are in BCAI format), switch modes with the **change_mode** command. Then enter the position of each flaw. In sector mode, enter the cylinder, head, and sector where the flaw exists. In BCAI mode, enter the cylinder, head, byte-count-after-index, and bit-length.

To enter the defect data from one or more files, select each file by typing

file *filename*

where *filename* is replaced with the actual name of the file. The first line of the file defines the drive type and looks like the following:

n *drivename*

where *drivename* is the drivename which can be found in the file */mnt/bin/lib/DB_diskfmt* on the Service Processor (for example, DKD-005 for the 1/2 GByte NEC 2352 drive). Following the drive name line, the file has remaining lines that contain either BCAI inputs or logical sector inputs. A BCAI input is preceded with the letter **d** while a logical sector input is preceded with the letter **s**. A spare sector can be specified by using a logical sector number one greater than the last used sector number. To specify a track that is to be mapped to an alternate track, use the letter **m** followed by the cylinder and head of the bad track. One or more spaces separate each item in the file and a new line can be started between any two items. Comments may be included in the file by typing a **#** at any place on a line. All characters from the **#** to the end of the line are ignored. Blank lines are also allowed. A file containing defect data is shown in the following figure:

Figure 4–12, Creating a Defect Data File

```
#------------------------------ start of file ----------------------------
# serial number: 003013
n  DKD-005  # NEC 2352 drive
#   cyl  hd   bcai  len       cyl  hd   bcai  len
d   157  15  24395   34    d  297  14  14102    4  # BCAI Inputs
d   466   0    206    1

#   cyl  hd  sec             cyl  hd  sec
s   457  12   42           s  821   5   11         # Logical sector inputs
#------------------------------ end of file ----------------------------
```

NOTE

To abort *dev4110* while in this subtest, press CTRL and **b**. The SPU takes about 30 seconds to dump memory. Then the SPU prompt is displayed. If CTRL and **c** are pressed, it is intercepted by the subtest and the subtest prompt is displayed.

The current input mode (logical sector or defect map mode) is revealed in the slip prompt. If in logical sector input mode, the prompt appears as follows:

```
make_bad (D1;Sector)->
```

If in defect map input mode, the prompt appears as follows:

```
make_bad (D1;Defect)->
```

When a defect is entered, a list of logical sectors that correspond to the defect is generated as follows:

Figure 4–13, List of Logical Sectors For a Defect

```
make_bad (D1;Defect)-> 293 1 572 9510
        Associated logical sectors: 59 0 1
```

The previous figure illustrates that three sectors are affected by the media flaw which starts at byte 572 from index and is 9510 bits long.

The byte before and after any flaw is also considered bad to avoid the possibility that a flaw might be slightly bigger than is reported in the defect map.

If two defects affect the same sector, the logical sector is only entered in the *INPUTS THAT WILL BE SLIPPED* input list once. Subsequent entries of the same sector are entered in the *INPUTS THAT WILL NOT BE SLIPPED* list.

For instance, if the following two defects exist:

cylinder	head	bcai	bit-length
10	0	70	1
10	0	80	1

Both defects fall in sector 0 so when the second defect is entered, it does not create a new entry in the input list. This is shown in the following figure:

Figure 4–14, Two Defects Entered for the Same Sector

```
make_bad (D1;Defect)-> 10 0 70 1
     Associated logical sectors: 0
make_bad (D1;Defect)-> 10 0 80 1
     Associated logical sectors: (0)
```

The following is an example of entering a bad location and deleting it:

Figure 4–15, Entering and Correcting An Incorrect Location

```
make_bad (D1;Sector)-> 17 7 23
make_bad (D1;Sector)-> 17 7 46
make_bad (D1;Sector)-> d 17 7 46
        Input 17 7 46 has been removed
make_bad (D1;Sector)-> list

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
        ---------------------- no sectors ----------------------
   .

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
 17  7  23
```

The following is an example of switching drives and entering defects for the drive from a file:

Figure 4–16, Entering Defects for a New Disk Drive

```
make_bad (D1;Sector)-> dr 2
        Drive 2 is now selected
make_bad (D2;Sector)-> !cat SN12345678          <-to illustrate unix cmd
# Serial Number: 12345678
d  100  5 10536     1
d  257  9 26401  1000
> file SN12345678                                <-prompt after unix cmd is '>'
make_bad (D2;Sector)-> list

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
         ---------------------- no sectors ----------------------

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
 100  5  11  10536     1      N/A USR | 257  9  35  26401  1000      N/A USR
 257  9  36  26401  1000      N/A USR |

make_bad (D2;Sector)->
```

In the above example, the manual mode is set to `Sector` but the file contains BCAI information. This is does not create a problem. The manual mode only applies to manual inputs. File inputs are controlled by the character that precedes the defect location. The defect at cylinder 257, head 9 was large enough to overlap two sectors so both were flagged bad.

Type **quit** after entering all inputs to exit the subtest. Then Subtest 101 starts.

4.5.1.2 Subtest 101, System Format of SMD

Subtest 101 formats a drive for system use except for the creation of the diagnostic cylinder which is performed in Subtest 102, Initialize Diagnostic Cylinder. This subtest consists of the following five parts:

1. Format with no spare sectors to allow pattern test of all sectors.

2. Pattern test by first reading the `0x00000000` pattern which was written during format. Then write and read seven other patterns which are shown below:

   `0xAAAAAAAA` `0xAF5BAF5B`
   `0xFFFFFFFF` `0x5EB75EB7`
   `0xEBD6EBD6` `0x6DB66DB6`
   `0xD7ADD7AD`

   These patterns can be replaced by up to 16 other patterns if the file *dev4110.patt* is created in the current directory or if it is created in */mnt/bin/lib*. The current directory is searched first. The format of the file is free form with eight hex characters per pattern and at least one space separating the patterns. More than one line can be used to define the patterns. Blank lines are also allowed.

Any sector-related errors (header not found; hard ECC error; or correctable ECC error) are retried immediately and, if an error occurs again in ten retries (does not have to be the same error), the sector location is saved for later slipping. A maximum of 1365 errors can be saved per drive. The maximum includes the flaws from Subtest 100, Input Defects Before Formatting. Errors that do not repeat are discarded. If a verbose value of 2 is selected, each error is printed at the time it is detected.

3. Format with one spare sector per track.

4. Slip sectors which were input in Subtest 100, Input Defects Before Formatting or which were detected as bad during pattern test. If more than one sector is bad on a track, one sector is slipped and then the remaining bad sectors are added to the bad block table. This table associates the bad sectors with good sectors located in the last and possibly next to the last cylinder. Up to 639 sectors can be relocated.

   If a verbose value of 1 (default) is selected in the test parameter menu, a summary of bad sectors for each drive is printed before slipping occurs.

5. The last operation is to write five copies of the bad block table to the last 25 sectors of the disk.

4.5.1.3 Subtest 102, Initialize Diagnostic Cylinder

Subtest 102 writes special patterns to the diagnostic cylinder which is the cylinder just before the cylinder used for relocated sectors. Sector 0 of each track contains a table of contents that describes what is written to each of the remaining sectors. If no sectors have been relocated on a track, sector 1 is marked bad. The next twelve sectors contain Error Correction Code (ECC) errors of length 1 to 12 respectively. The remaining sectors have the seven patterns listed in Subtest 101, System Format of SMD written to them. Sector 13 (or 14 if sector 1 is marked bad) will contain an `0xAAAAAAAA` pattern, sector 14 (or 15) will contain an `0xFFFFFFFF` pattern, and so on with the patterns repeating every seven sectors. Refer to the section Disk Parameters File, *DB_diskfmt* Description for more information about the diagnostic cylinder.

4.5.1.4 Subtest 103, Verify System Format of SMD

Subtest 103 reads every logical sector in the usable portion of the disk. The usable portion of each drive is all the cylinders preceding the diagnostic cylinder and all relocated sectors (which are found on the innermost one or two cylinders). When a header not found error occurs, the bad block table is checked to see if the header has been relocated. If it has been relocated, the alternate sector is read.

Any errors other than those due to sector relocation are reported and count against the device errors per subtest limit. If the number of errors exceeds the limit, the drive is failed.

4.5.1.5 Subtest 104, Test Diagnostic Cylinder

Subtest 104 reads each sector on the diagnostic cylinder and expects soft or hard ECC errors on sectors 1 through 12 of each track (or sectors 2 through 13 if sector 1 is marked bad). The remaining sectors must be error-free.

4.5.2 Class 2 Subtest

The Class 2 subtest performs interactive testing of all selected drives. The Class 2 subtest can only be started by explicitly using the **-c** or **-s** options when *dev4110* is invoked. The Class 2 subtest is shown in the following table.

Table 4–7, Class 2 Subtest

SUBTEST	DESCRIPTION	TIME (min:sec)
200	Interactive Test	N/A

4.5.2.1 Subtest 200, Interactive Test

Subtest 200 provides a special interactive test interface to perform the following disk maintenance tasks:

- Pattern test portions of the disk
- Slip bad sectors
- Format single tracks
- Several more useful operations

A help facility is available when needed.

> **CAUTION**
>
> Three of the interactive test commands are potentially data destructive: **format**, **pattern_test**, and **slip_sectors**.

Each command performs an automatic save of the data it is to overwrite before the operation begins. However, the commands can still be data destructive if one of the following actions is attempted:

1. An attempt to pattern test more than one track is performed and **y** is entered when asked to confirm multiple track pattern test (because only one track can be saved).

2. An attempt to preserve a track's data before a destructive operation fails and a response is entered to force the operation anyway.

3. The pattern test, slip sectors, or format operation has completed and then data errors occur during the restore of the track's data.

4. The keys CTRL and **c** or CTRL **b** are pressed in the middle of the operation.

5. Automatic save is disabled by using the **toggle_save** command.

6. A power outage or some other unforeseen event causes the test to abort in the middle of the operation.

4.5.2.2 Interactive Test Invocation

To invoke the interactive test, use the procedure shown in the following figure:

Figure 4–17, Interactive Test Invocation Sequence

```
(spu)> cd /mnt/test
(spu)> mminit -s
(spu)> dshell
: test dev4110 -c 2 (or test dev4110 -s 200)
```

4.5.2.3 Interactive Test Parameter Menu

The prompts are displayed as shown in the following figure. Use defaults for the first four prompts and then select the drives to test and press RETURN after each drive is selected. After all drives have been selected, press RETURN twice and enter **y**. After about a minute, the interactive test will start.

The following figure shows the prompts for the Interactive Test Parameter Menu:

Figure 4–18, Interactive Test Parameter Menu

```
                        ENTER TEST PARAMETERS
          []      Encloses allowed input ranges or values
          ()      Encloses the default value
          ^       Returns to the previous prompt
          :nn     Returns to the prompt # nn
          :       Returns to the first unsatisfied prompt

          ?       Prints an additional help menu
 1:  Number of errors allowed per device each subtest
    [0-65535]                                          (0) -> RETURN
 2:  Number of devices failed before test aborts
    [0-65535]                                         (12) -> RETURN
 3:  Verbosity of test [0-65535]                       (3) -> RETURN
 4:  Ioconfig file                              (/ioconfig) -> RETURN

                PERIPHERAL CONFIGURATION DATA[1]
     IOP  MBUS  TYPE       CSR    INT  UNIT  TYPE
     ---  ----  --------  -----   ---  ----  -------
 1)   3     0   DKC-001   0xdc0    1     0   DKD-001
 2)   3     0   DKC-001   0xdc8    4     0   DKD-001
 3)   6     1   DKC-001   0x3f0    3     0   DKD-001

Enter device 99 to begin user-defined configurations or 0 to end selection

 5:  Device selection [0,1-3,99]                       (0) -> 1 RETURN
 6:  Device selection [0,1-3,99]                       (0) -> 2 RETURN
 7:  Device selection [0,1-3,99]                       (0) -> 3 RETURN
 8:  Device selection [0,1-3,99]                       (0) -> RETURN
 9:  Enter OK, or :NN to return to question NN [OK]
                                                      (OK) -> RETURN

   ********** WARNING! **********
THIS TEST IS POTENTIALLY DISK DATA DESTRUCTIVE!
IF YOU CONTINUE, DATA COULD BE LOST!
Do you want to continue [yn]      -> y RETURN
```

[1] This information is only displayed once; to display it again , type **i** RETURN at any prompt during test parameter query.

4.5.2.4 Interactive Test Mode

After the test parameter menu prompts have been answered the following is displayed indicating interactive test mode:

> **NOTE**
>
> In the following figure, D1 indicates drive 1 is selected and Save indicates that track data is saved during the data destructive commands **format**, **pattern_test**, and **slip_sectors**.

Figure 4–19, Interactive Test Mode

```
                    INTERACTIVE TEST MODE
                       FOR SMD DRIVES

Type 'h' for help
(D1;Save)->
```

Type **h** to list the available interactive test commands. For additional help, refer to the command descriptions in the following section. Only a unique portion of the beginning of a command needs to be typed.

To exit the interactive test, type **q** and press RETURN. If a track is saved, a prompt is displayed indicating whether data can be lost. If **y** is entered or if a track is not saved, *dev4110* ends normally and exits back to the dshell prompt. If **n** is entered, the test returns to the interactive prompt without executing changes.

4.5.3 Interactive Test Commands

The following table lists the available commands and also a description of each command.

Table 4–8, Interactive Test Commands

COMMAND	DESCRIPTION
a[lternate_seek]	Seek between two selected cylinders
b[bt_display]	Display Bad Block Table
d[rive_select]	Select a drive
f[ormat]	Reformat one track
hd[e_display]	Display header, data, and ECC
h[elp]	Display list of commands and their arguments
i[nitialize_bbt]	Rebuild damaged Bad Block Table
p[attern_test]	Pattern test range of sectors
q[uit]	Exit from the Interactive Test
r[estore_track_data]	Restore previously saved track data
sa[ve_track_data]	Save one track's data
sc[an_flaws]	Produce file of all marked-bad sectors
se[ctor_display]	Display data from range of sectors
sl[ip_sectors]	Slip one or more sectors
st[atus]	Display current drive and which track is saved
to[ggle_save]	Switches between allowing and disabling auto-save
tr[ack_headers_display]	Display track headers
v[erify_format]	Read a range of sectors
!UNIX-command	Execute UNIX command
'comment	Any comment; ignored

4.5.4 Interactive Test Commands Descriptions

The following sections describe the interactive test commands.

4.5.4.1 Seek Between Two Selected Cylinders

a[lternate_seek]

This command performs seeks between two selected cylinders.

4.5.4.2 Display Bad Block Table

b[bt_display]

This command displays the active drive's bad block table as follows:

Figure 4–20, Active Drive's Bad Block Table Display Screen

```
BAD BLOCK TABLE FOR ccu/mb/csr/d=3/0/3f0/0

        CYL HD SEC  CYL HD SEC  CYL HD SEC  CYL HD SEC  CYL HD SEC
        275  3  48  387 11  48
```

When one bad sector is slipped on a track, the bad sector is slipped and all subsequent sector headers are shifted by one with the last logical sector replacing the spare sector. Additional slipped sectors on a track causes the associated headers to be marked bad and the sectors to be relocated. To display which sectors are bad, use the **track_headers_display** command. The bad sectors are flagged with a *B. To display a different drive's bad block table, use the **drive_select** command to change drives.

4.5.4.3 Select a Drive

d[rive_select] [*drive-number*]

This command can change which drive is the active drive. Most of the other commands operate on the active drive only. The argument *drive-number* is the entry number from the Drive Configuration Data which was displayed at the end of the user prompts. To display the entry numbers, enter **d** and press RETURN. The Drive Configuration Data and active drive are then redisplayed, and a prompt is displayed to enter a new drive entry number. Enter **0** or press RETURN for the current drive.

Use of the **drive_select** command is shown in the following figure:

Figure 4–21, Drive Configuration Data Screen

```
(D1;save)-> d

                    DRIVE CONFIGURATION DATA

    IOP MBus MBus  Int   Unit  #    #  Phys  Log
     #   #   CSR  Level   #   Cyl  Hds  Sec  Sec  Name
    --- --- ----- -----  ---  ---  ---  ---  ---  -------
 1.  3   0  0x3f0   2     0   760   19   60   59  DKD-005
 2.  3   0  0x3f0   2     1   760   19   60   59  DKD-005
 3.  3   0  0x3f8   3     0   842   20   46   45  DKD-001

        Drive 1 is currently selected
        Enter new selection or 0 to leave it as is (0,1-3) [0] -> 2
        Drive 2 is now selected
(D2;save)-> d 3
        Drive 3 is now selected
(D3;save)->
```

The configuration data does not display if the drive number is entered. Also, the drive number is redisplayed with each prompt as D*x* where *x* is the current drive number.

4.5.4.4 Reformat One Track

f[ormat] *cyl hd*

This command unslips sectors or recreates a bad track's headers. If automatic save is enabled (the prompt indicates Save), then before the track is reformatted, an attempt is made to read every logical sector (including relocated sectors). If a read fails, the options of retrying the sector, terminating the format before any destructive operation occurs, or forcing the format even though one or more sectors may not be recovered are displayed.

> **CAUTION**
>
> If auto save is disabled (Nosave mode) by using the **toggle_save** command, the track is reformatted without preserving the track's data.

In the following figure, cylinder 300, head 12 is reformatted on an NEC drive (760 cylinders, 19 heads, 59 logical sectors). The track contains two slipped sectors as shown. The second bad sector has been relocated.

> **CAUTION**
>
> If auto save is enabled and sectors are unslipped by the **format** command, data is restored to the bad sectors which could cause the data to no longer be readable. This is okay since the data is still saved. Slip the bad sectors using the **slip_sectors** command and then use the **restore_track_data** command to restore the track's data.

The following figure shows how to perform the restoration operation:

Figure 4–22, Formatting a Track

```
(D3;Save)-> format 300 12
        No data has been previously saved. Saving this track
        Data save operation complete

  TRACK HEADERS BEFORE FORMAT:
 cylinder = 300  head = 12
 47 48 49 50 51 52 53 54 55 56 57 58  0  1  2  3 *B  4  5  6  7  8  9 10 11
 12 13 14 *B 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
 37 38 39 40 41 42 43 44 45 46

        About to format cylinder 300  head 12. Are you sure? [yn] -> y
        Format complete.  2 sectors have been unslipped.
        Restoring data to track
        Data restore operation complete
(D3;Save)-> track 300 12
 cylinder = 300  head = 12
 48 49 50 51 52 53 54 55 56 57 58 *S  0  1  2  3  4  5  6  7  8  9 10 11 12
 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
 38 39 40 41 42 43 44 45 46 47
(D3;Save)->
```

Then pattern test the track and identify which sectors are bad, and slip the bad sectors. Then restore the original track data. These operations are shown in the following figure:

Figure 4–23, Restoring the Original Track Data

```
(D3;Save)-> patt 10 times 300 12 0 to 300 12 58
     Patterns used:
      aaaaaaaa ffffffff ebd6ebd6 d7add7ad af5baf5b 5eb75eb7 6db66db6
        If you save this track, you will lose the saved data for
        drive 3: cylinder 300  head 12.
        Save data? [yn] -> n                <- previous data is what you want
        WARNING - DATA NOT SAVED

 pass # 1
dev4110 ERROR (0x1e) ccu/mb/csr/d 3/0/3f0/0 - Correctable ECC error
 cyl: 300 hd: 12 sec: 4

dev4110 ERROR (0x06) ccu/mb/csr/d 3/0/3f0/0 - Hard ECC error
 cyl: 300 hd: 12 sec: 16

---- more errors will be detected as the test loops. ----

(D3;Save)-> slip
slip (D3;Save;Sector)-> 300 12 4
slip (D3;Save;Sector)-> 300 12 16
slip (D3;Save;Sector)-> e

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
         ---------------------- no sectors ----------------------

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
 300 12   4                  N/A USR | 300 12  16                   N/A USR

        Are you sure inputs are ready for slipping? [yn] -> y
      sector 300 12  4 and
             300 12 16: slipped
(D3;Save)-> restore
        Data has been saved for drive 3: cylinder 300 head 12
        Do you really want to restore this data? [yn] -> y
        Data restore operation complete
(D3;Save)->
```

4.5.4.5 Display Header, Data, and ECC

hd[e_display] *cyl hd sec* [**to** *cyl hd sec*]

This command is used to display the sector data, the header, and ECC. The data is read in raw mode; therefore, the controller does not perform error checking.

> **NOTE**
>
> The sector specified for the **hde_display** command is the physical sector numbered from index. All other commands search all headers on the track until the requested header is found.

To illustrate the **hde_display** command, the track headers displayed by the **track** command and the sector displayed by the **hde_display** command are shown in the following figure:

Figure 4–24, *hde_display* Command

```
(D2;Save)-> track 27 5
 cylinder =  27  head =  5
 55 56 57 58 *S  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
 45 46 47 48 49 50 51 52 53 54
(D2;Save)-> hde 27 5 0

Phys:  27  5   0  Log:  27  5  55  Type: 0        Header:001b0537 ECC:2bc18e26
000  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
020  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
040  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
060  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
080  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0a0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0c0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0e0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
100  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
120  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
140  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
160  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
180  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1a0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1c0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1e0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6

(D2;Save)->
```

The previous figure shows that the physical sector 0 corresponds to logical sector 55 on this track.

4.5.4.6 Display List of Commands and Arguments

h[elp]

This command lists the commands and their arguments.

4.5.4.7 Rebuild Damaged Bad Block Table

i[nitialize_bbt]

This command reads the Bad Block Table (BBT) and attempts the following:

1. Tries to read (successfully)any one of the five copies in the BBT
2. If any one of the copies is read successfully, it is used to recreate all of the other copies
3. If none of the copies in the BBT are read successfully, a message is printed out stating that all copies of the BBT are bad. The following prompt is then printed:

```
Do you want to retry the last operation [yn] ->
```

If this prompt is answered with **y**, program will attempt to read (successfully) the BBT again. If **n** if entered, the following prompt will be displayed:

```
WARNING:  This could result in several NEW Header Not Found Errors.

Do you want to continue [yn] ->
```

> **WARNING**
>
> If the previous prompt is answered with **y**, any previous entries to the BBT will be erased. Run *verify_format* and take appropriate action if any Header Not Found messages appear.

The following figure displays an example of this command:

Figure 4–25, Initialize_bbt Example

```
(D1;Save)-> initialize_bbt
Drive 1: BBT copy 1 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 2 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 3 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 4 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 5 is bad.  Count field is 2779096485. Should be 0-638.


dev4110 ERROR (0x4d)-ccu/mb/csr/d=7/3f0/0 All copies of BBT are bad
The internal copy of the BBT is cleared since no good copies were found
Do you want to retry the last operation [yn] -> y
Drive 1: BBT copy 1 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 2 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 3 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 4 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 5 is bad.  Count field is 2779096485. Should be 0-638.


dev4110 ERROR (0x4d)-ccu/mb/csr/d=7/3f0/0 All copies of BBT are bad
The internal copy of the BBT is cleared since no good copies were found
Do you want to retry the last operation [yn] -> n

WARNING: This could result in several NEW Header Not Found Errors.

Do you want to continue [yn] -> y
Done
```

4.5.4.8 Pattern Test a Range of Sectors

p[attern test] [*nnn* **times**] *cyl hd sec* [**to** *cyl hd sec*] [**with** *pat1* [*pat2...*]]

This command pattern tests the selected sectors repetitively. It is used to test part or all of the system portion of the disk (all cylinders before the diagnostic cylinder and relocated sectors). If the test is constrained to one track, the data is automatically saved and restored when the operation is completed. If more than one track is selected, a warning prompt is displayed and the operation can be terminated at that point to prevent permanent overwrite of data. The automatic save can be disabled with the **toggle_save** command. The argument [**to** *cyl hd sec*] can be substituted with the argument [**to** *end*].

Arguments:

- [*nnn* **times**]— Repetition count. possible responses include:
  - **1**—Lower bound value, default value.
  - **2,147,483,647**—Upper bound value.
- *cyl hd sec*— Starting sector.
- [**to** *cyl hd sec*] Ending sector, default is starting sector.
- [**with** *pat1*[*pat2...*]]— Up to 16 4-byte hex patterns can be specified. The default is the following seven patterns: `aaaaaaaa ffffffff ebd6ebd6 d7add7ad af5baf5b 5eb75eb7 6db66db6`

4.5.4.9 Restore Previously Saved Track Data

r[estore_track_data]

This command can restore data from the general buffer to the original track (the track where the data was originally stored). The **save_track_data** command stores data from a selected track into the general buffer. In addition, the **format** and **pattern_test** commands save into the general buffer before their operation and automatically restore the data afterward. The data remains in the buffer so it can be restored again at a later time if, for instance, the first restore was bad.

If data is saved and not restored and then an attempt is made to quit the interactive test, a message is displayed indicating there is saved data and a prompt is displayed that allows reentering the interactive test or exiting the test.

The following figure shows an example of using the restore command:

Figure 4–26, Restore Command Example

```
(D3;Save)-> restore
        Data has been saved for drive 3: cylinder 273  head 17
        Do you really want to restore this data? [yn] -> y
        Data restore operation complete
(D3;Save)->
```

If **n** is entered, the data is not restored, the operation is terminated, and the test returns to the interactive test prompt.

If the current drive has been changed from drive 3 to drive 9 since the last save operation, the following is displayed:

Figure 4–27, Changing Drives Since Last Save Operation

```
(D9;Save)-> restore
        Data has been saved for drive 3: cylinder 273  head 17
        However, drive 9 is currently selected.
        Use 'd[rive_select]' to change drives and try again.
(D9;Save)->
```

If data was not saved before attempting to restore the data, the following is displayed:

Figure 4–28, Attempting to Restore Data Before Saving

```
        No data has been previously saved so restore aborted.
(D9;Save)->
```

4.5.4.10 Save One Track of Data

sa[ve_track_data] *cyl hd*

This command saves one track of data into the general buffer. The data can be restored with the **restore_track_data** command. Data on relocated sectors is also saved so, if the track is reformatted (which removes the relocated sectors), all the track's data can be restored.

> **NOTE**
>
> The prompt in the following figure specifies `Nosave`. This indicates the command **toggle_save** has previously been entered so saves are not automatic on destructive operations. It has no affect on this command. Normally, `Save` appears since disabling automatic save can be data destructive.

The following is an example of saving one track:

Figure 4–29, Saving One Track

```
(D4;Nosave)-> save 525 5
        Data has already been saved for drive 4: cyl=601, hd=14
        Are you sure you want to overwrite this data? [yn] -> y
        Data save operation complete
(D4;Nosave)->
```

4.5.4.11 Produce File of All Bad Sectors

sc[an_flaws] *filename*

This command reads every header on the disk looking for headers marked bad or which contain invalid information. Each of these headers produces an entry in the ASCII file entered on the command line. The file is in the same format as is read by the **file** command in Subtest 100, Input Defects Before Formatting. Refer to Subtest 100 for a description of the file format.

4.5.4.12 Display Data From Range of Sectors

se[ctor_display] *cyl hd sec* [**to** *cyl hd sec*]

This command displays the data from one or more consecutive sectors.

The following figure shows an example of the **sector_display** command:

Figure 4–30, Displaying Sector Data

```
(D1;Save)-> sector 27 5 55

        Sector  27  5 55:
000  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
020  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
040  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
060  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
080  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0a0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0c0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0e0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
100  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
120  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
140  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
160  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
180  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1a0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1c0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1e0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6

(D1;Save)->
```

This example is almost identical to the example for the **hde_display** command. The major difference in the two methods is that with the **sector_display** command, the read is sensitive to data errors. In the raw read mode that the **hde_display** command uses, no data errors are reported. If a data error does occur while executing the **sector_display** command, it is retried 10 times and any successful read results in the sector being displayed without an error message. If ten unsuccessful attempts to read the sector occur, then an error message is displayed and a prompt is displayed whether to retry, force, or skip the read.

To retry the read another 10 times press (RETURN). To retry the read a different number of times, enter a number. From 1 to 2,147,483,647 retries can be specified. Every five retries a seek is alternately performed one cylinder in and back or one cylinder out and back.

If **force** is entered for the read, the buffer the data was supposed to be read into is displayed. If an ECC error occurred, the data is whatever was read. If some other error occurred, the data may be whatever was in the buffer before the read and be invalid. The error reported is the last error that occurred; it can be used to determine if the data is valid.

> **NOTE**
>
> At present, the software does not correct the data if it is a correctable ECC error.

If **skip** is entered, the sector is skipped. If there are more sectors to read in the range specified, then the display continues with the next sector.

The following is an example of a read that fails:

Figure 4–31, Failed Read Example

```
(D1;Save)-> sector 400 22 18 to 400 22 19

dev4110 ERROR (0x06) ccu/mb/csr/d 3/0/3f0/1 - Hard ECC Error
 cyl:400 hd:22 sec:18

                        10 attempts to read sector failed
                        what to do [# of retries/force/skip] (10) -> 100

dev4110 ERROR (0x06) ccu/mb/csr/d 3/0/3f0/1 - Hard ECC Error
 cyl:400 hd:22 sec:18

                        100 attempts to read sector failed
                        what to do [# of retries/force/skip] (100) -> skip

        Sector 400 22 19
000  00000000 00000000 00000000 ff300020  1023eee10 99903220 b2d3d4d1 ab222290
020                     ------------ and so on -------------
```

Sector 18 was not displayed since it was skipped. What was displayed was the next sector, sector 19 which was read successful. The next sector (sector 19) was displayed because it was read successfully in one of the first 10 attempts.

4.5.4.13 Slip One or More Sectors

sl[ip_sectors]

> **CAUTION**
>
> Read the caution and following example at the end of this command's description before using this command.

This command is used to slip sectors which are generating errors or to enter defect map information to **slip sectors**. If automatic save of track data is enabled (indicated by `Save` in the prompt), an attempt to save and restore all sectors is made to minimize data loss unless automatic save is disabled with the **toggle_save** command.

Entering the **slip_sectors** command starts a separate command processor which expects bad sectors or defect map information to be entered. A help facility which lists the available commands is displayed if **h** is entered. The help output appears as follows:

Figure 4–32, *slip_sectors* Command Help Screen

```
SLIP_SECTORS commands:
c[hange_mode]       - toggles logical sector/defect map entry mode
cyl hd sec          - logical sector to be slipped
cyl hd bcai len     - defect map entry to be slipped
d[elete] cyl hd sec - delete logical sector entry
d[elete] cyl hd bcai len - delete defect map entry
e[xecute]           - slip the sectors now
h[elp]              - display this information
l[ist]              - list inputs made so far
q[uit]              - exit from 'slip_sectors'
!unix-command       - execute unix command
'comment            - comment; ignored
```

To slip sectors which are generating errors, the normal usage is to display the track headers for each track on which a sector is to be slipped by using the **track_headers_display** command. This command displays the headers before they are altered so the slip can be verified. Then enter all sectors which need to be slipped, verify the entries using the **list** command, delete any incorrect entries with the **delete** command, and then begin the slip with the **execute** command. New sectors can be entered at any time until the **execute** command is completed. As a safeguard, when the **execute** command is entered, the list of sectors is automatically displayed and a prompt is displayed to confirm the list. If **n** is entered, execution returns to the **slip_sectors** prompt with all inputs still intact.

To exit the test at any time enter the **quit** command.

To enter defect map information, first enter the **change_mode** command and then follow the above procedure. It may not be desirable to display track headers if there is a large number of tracks.

> **NOTE**
>
> When entering the defect map for a new disk, a disk that has just been formatted and does not yet contain a file system, the automatic save may be disabled with the **toggle_save** command before entering the **slip_sectors** command. This causes the slip to proceed much faster.

CAUTION

If the **toggle_save** command is executed prior to the slip, any usable data that exists on the disk will be lost, so use this approach with caution.

The current input mode (logical sector or defect map mode) is revealed in the slip prompt. If logical sector input mode is selected, the prompt appears as follows:

```
slip (D1;Save;Sector)->
```

If defect map input mode is selected, the prompt appears as follows:

```
slip (D1;Save;Defect)->
```

When a defect is entered, the associated track's headers are immediately read and a list of logical sectors that correspond to the defect are listed as shown in the following figure:

Figure 4–33, Entering a Defect

```
slip (D1;Save;Defect)-> 293 1 572 9510
        Associated logical sectors: *S 0 1
```

The previous figure illustrates that three sectors are affected by the media flaw which starts at byte 572 from index and is 9510 bits long. The *S indicates that one of the affected sectors is the spare sector. If a *B is displayed, then the media defect affects an already bad sector which is not reslipped.

The byte before and after any flaw is also considered bad to avoid the possibility that a flaw might be slightly larger than is reported in the defect map.

If two defects affect the same sector, the logical sector is only entered in the *INPUTS THAT WILL BE SLIPPED* input list once. Subsequent entries of the same sector are entered in the *INPUTS THAT WILL NOT BE SLIPPED* list. For example, the following two defects exist:

cylinder	head	bcai	bit-length
10	0	70	1
10	0	80	1

Both defects are in sector 0 so when the second defect is entered, the sector is not slipped again. The slipping of a sector with two defects is shown in the following figure:

Figure 4–34, Slipping Sectors With Two Defects

```
slip (D1;Save;Defect)-> 10 0 70 1
        Associated logical sectors: 0
slip (D1;Save;Defect)-> 10 0 80 1
        Associated logical sectors: (0)
        Sectors in () have already been input and will not result in a new
        entry. A (*G) means the flaw falls entirely in a gap and is ignored.
slip (D1;Save;Defect)-> list

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
  10  0   0     80     1      N/A USR |

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
  10  0   0     70     1      N/A USR |
```

The above example also refers to a *gap* which is listed in the Associated logical sectors list as a (*G). The *G indicates that the entire media flaw was in a space between sectors or located after the last sector on the track.

NOTE

In the current version of the *dev4110* test, no bytes are considered part of a gap. A flaw in any byte results in one or more sectors being slipped.

When the slip is executed, the sectors are slipped one track at a time. If automatic save is active, the data for the track is copied to a special buffer (not the general buffer used by the **save_track_data** command) before the slip and is restored after the slip. One problem exists which is that correctable ECC errors are not corrected so a correctable ECC error may not be fixable without a loss of data.

To verify the slip, display the track headers once more. All slipped sectors are marked with a ***B**.

An example of slipping the sectors 17 7 23, 17 7 36, 247 12 4, and 255 0 12 is shown in the following figure:

NOTE

Track 255 already has a slipped sector. Sector 12 that is about to be slipped *must* be a new error since sector 6 was slipped. Refer to the caution and following example at the end of the **slip_sectors** command description for more information about slipping two or more sectors on one track.

Figure 4–35, Display Slipped Track Headers

```
(D1;Save)-> track 17 7
cylinder =  17  head =  7
53 54 55 56 57 58 *S  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52
(D1;Save)-> track 247 12
cylinder = 247  head = 12
48 49 50 51 52 53 54 55 56 57 58 *S  0  1  2  3  4  5  6  7  8  9 10 11 12
13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
38 39 40 41 42 43 44 45 46 47
(D1;Save)-> track 255 0
cylinder = 255  head =  0
 0  1  2  3  4  5 *B  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56 57 58
```

If defect input mode is selected and an attempt is made to slip the sector on track 255 head 0 which has already been marked bad in the previous figure, the following figure shows what would be displayed:

Figure 4–36, Slipped Tracks Previously Marked Bad

```
slip (D1;Save;Defect)-> 255 0 3800 1
        Associated logical sectors: (*B-6i)
```

The associated logical sector has been previously marked bad (*B). The 6i indicates the sector's location from index. Starting with the first sector after index as 0, the sector marked bad is sector 6 after index (which is actually the seventh sector after index). This notation is necessary because, once a header is marked bad, the header no longer contains a logical sector number. The only known information is the header's position from index.

The following illustration shows how to delete incorrect **slip_sectors** inputs:

Figure 4–37, Deleting Incorrect *slip_sectors* Inputs

```
(D1;Save)-> slip
Type 'h' for help with slip commands
slip (D1;Save;Sector)-> 17 7 23
slip (D1;Save;Sector)-> 17 7 46
slip (D1;Save;Sector)-> d 17 7 46
       Sector  17  7 46 has been removed from the inputs
slip (D1;Save;Sector)-> list

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
         ---------------------- no sectors ----------------------

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
  17  7  23                   N/A USR |


slip (D1;Save;Sector)-> 17 7 36
slip (D1;Save;Sector)-> 247 12 4
slip (D1;Save;Sector)-> 255 0 12
slip (D1;Save;Sector)-> e

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
         ---------------------- no sectors ----------------------

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
  17  7  23                   N/A USR | 247 12   4                   N/A USR
  17  7  36                   N/A USR | 255  0  12                   N/A USR

       Are you sure inputs are ready for slipping? [yn] -> y

     sector  17  7 23 and
             17  7 36: slipped
     sector 247 12  4:
dev4110 ERROR (0x06) ccu/mb/csr/d 3/0/3f0/1 - Hard ECC error
 cyl:247 hd:12 sec:4

                      10 attempts to read sector failed
                      what to do [# of retries/force/skip] (10) -> 100

slipped
       sector 255  0 12: slipped
```

Sector 247 12 4 was difficult to read. After the first 10 attempts failed, a request was entered to retry 100 times. One of the 100 retries was successful so the sector slip operation was completed. If the 100 retries had been unsuccessful, a prompt to retry again, force continuation with the data that had been read from sector 4, or skip slipping this sector would have been displayed. If **skip** is selected, the remaining slips are still performed. Every five retries a seek is alternately performed one cylinder in and back or one cylinder out and back.

To verify the slips, display the track headers again as shown in the following figure:

Figure 4–38, Displaying the Track Headers

```
(D1;Save)-> track 17 7
cylinder =  17  head =  7
52 53 54 55 56 57 58  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
18 19 20 21 22 *B 23 24 25 26 27 28 29 30 31 32 33 34 *B 36 37 38 39 40 41
42 43 44 45 46 47 48 49 50 51
(D1;Save)-> track 247 12
cylinder = 247  head = 12
47 48 49 50 51 52 53 54 55 56 57 58  0  1  2  3 *B  4  5  6  7  8  9 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45 46
(D1;Save)-> track 255 0
cylinder = 255  head =  0
 0  1  2  3  4  5 *B  6  7  8  9 10 11 *B 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56 57 58
(D1;Save)->
```

Track 17 7 and track 255 0 now have only 58 logical sectors because one sector has been relocated from each track. The Bad Block Table may be displayed as shown in the following figure to verify the relocation actually occurred:

Figure 4–39, Displaying the Bad Block Table

```
(D1;Save)-> bbt

BAD BLOCK TABLE FOR ccu/mb/csr/d=3/0/3f0/0

         CYL HD SEC  CYL HD SEC  CYL HD SEC  CYL HD SEC  CYL HD SEC
          17  7  35  255  0  12

(D1;Save)->
```

If there had already been entries in the table, they would have been displayed also. This list of the Bad Block Table is a copy of the table in main memory. However, it does reflect what is on the disk since both the **slip_sectors** and **format** commands rewrite all five copies of the Bad Block Table before returning to the interactive prompt.

Execute the **verify_format** command for each of the tracks to verify the tracks.

CAUTION

The **slip_sectors** command has an inherent danger. If logical sectors are input to be slipped and two or more sectors are to be slipped on the same track, all the sectors that are on the same track must be slipped at one time since the entered sector numbers that are given to **slip_sectors** are logical numbers. **Slip_sectors** searches the headers on the track to find the sectors to slip. If a slip of one sector on a track is executed and then later a slip of another sector is executed (without reverifying the logical sector number of this second sector), the wrong sector might be slipped.

The following is an example to illustrate the preceding caution.

Cylinder 25, head 0 in sectors 8 and 12 is generating correctable ECC errors; therefore, the **track** command is executed as shown in the following figure to display the headers before the slip is performed:

Figure 4–40, Display Track Headers Prior to Slipping

```
(D1;Save)-> track 25 0
cylinder =  25  head =  0
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 *S
```

The following figure shows the headers that are displayed if only sector 8 is slipped:

Figure 4–41, Display Slipped Track Headers

```
(D1;Save)-> track 25 0
cylinder =  25  head =  0
 0  1  2  3  4  5  6  7 *B  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56 57 58
```

Sector 12 is no longer in the same place. The remaining bad sector has 11 as a header. If the **slip_sectors** command is reentered and sector 12 is slipped, correctable ECC errors will probably occur on sector 11 in the future.

If only sector 8 is slipped, the following two options are available:

1. Reformat the track which unslips sector 8 and then slip both sectors at one time. Display the track headers before the slip to verify only sector 8 has been previously slipped. If other sectors have been unslipped on this track in the past, slip them also.

2. Check the track headers and then slip the sector which is now in sector 12's place. In this example, slip sector 11.

If sector 12 has been incorrectly slipped, unslip the bad slip by performing option 1.

4.5.4.14 Display Current Drive and Saved Track

st[atus]

This command lists the Drive Configuration Data (for all selected drives) and displays which drive is the active drive. The **Status** command also displays which track was last saved in the general buffer by one of the commands: **format**, **save_track_data**, or **pattern_test**. In addition, a line is displayed which indicates whether automatic save of track data is enabled (`Save` is also shown in the prompt).

The following figure shows the information that is displayed when the **status** command is entered:

Figure 4–42, *status* Command

```
(D1;Save)-> status

                    DRIVE CONFIGURATION DATA

    IOP MBus MBus  Int   Unit  #    #   Phys  Log
     #   #   CSR  Level   #   Cyl  Hds  Sec  Sec  Name
    --- --- ----- -----  ---  ---  ---  ---  ---  -------
 1.  3   0  0x3f0   2     0   760   19   60   59  DKD-005
 2.  3   0  0x3f0   2     1   760   19   60   59  DKD-005
 3.  3   0  0x3f8   3     0   842   20   46   45  DKD-001

        Drive 1 is currently selected

Saved Track:  drive 3  cylinder 300 head 12

Save of track data is automatic during destructive commands.
(D1;Save)->
```

4.5.4.15 Switch Between Enable and Disable of Automatic Save

to[ggle_save]

This command switches between enable and disable of automatic save. Automatic save occurs during the commands **format**, **pattern_test**, and **slip_sectors**. The displayed interactive prompt indicates whether automatic save is enabled (`Save`) or disabled (`Nosave`). Also, the **Status** command displays the current setting of automatic save.

4.5.4.16 Display Sector Numbers

tr[ack_headers_display] *cyl hd* [**to** *cyl hd*]

This command displays the sector numbers for each of the sectors on a track. The first sector number is located physically just after the index so the number is not always zero since sectors are skewed depending on the head number. For example, on head 0, logical sector 0 is the first sector after index, and on head 1, logical sector 0 is the second sector.

Each track has one spare sector (unless it is used because one or more sectors were slipped on the track). The spare sector is marked ***S**. If any sectors have been flagged as bad, they are marked ***B**.

The following figure shows the display of sector numbers for the sectors and a ***S** for a spare sector on a track:

Figure 4–43, Displaying Sector Numbers and Spare Sector

```
(D1;Save)-> track 427 2
cylinder = 427  head =  2
58 *S  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55 56 57
(D1;Save)->
```

In the previous figure, sector 0 is actually the third sector on the track since it is head 2. Also the spare sector (***S**) is just before sector 0.

The following figure shows the display of sector numbers for the sectors and a ***B** for a bad sector on a track:

Figure 4–44, Displaying Sector Numbers and Bad Sector

```
(D1;Save)-> track 208 12
cylinder = 208  head = 12
47 48 49 50 51 52 53 54 55 56 57 58  0  1  2  3  4  5 *B  6  7  8  9 10 11
12 13 14 15 16 17 18 19 20 21 22 *B 24 25 26 27 28 *B 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45 46
(D1;Save)->
```

In previous figure, three sectors have been flagged bad. The spare sector has been used and sectors 23 and 29 have been relocated and appear in the Bad Block Table (refer to the **bad_block_table** command for more information). Also sector 0 is the 13th sector on the track because this is head 12.

4.5.4.17 Read a Range of Sectors

v[erify_format] [*nnn* **times**] *cyl hd sec* [**to** *cyl hd sec*]

This command allows repetitive reads of a range of sectors to attempt to detect read errors. The command is not data destructive so it can be used on a customer's disks. It does not perform a data compare since the data is unknown. As the verify progresses, a pass count is incremented on the display to indicate how far the execution has progressed. Any errors are displayed without retry and the reads continue. If any sectors have been relocated, the relocated sectors are read. The argument [**toR** *cyl hd sec*] can be substituted with [**to** *end*]

4.5.4.18 Execute UNIX Command

! *UNIX-command*

This command is used to issue UNIX commands. After the UNIX command is completed, execution is returned to the interactive prompt.

The following figure shows an example of entering the **!** interactive test command and the **ps** UNIX command:

Figure 4–45, Issuing a UNIX Command

```
(D1;Save)-> !ps
   PID TTY TIME CMD
     2 co  0:08 -
   273 co  0:00 dshell
   274 co  0:09 dev4110.t -c 2
   281 co  0:00 sh -c ps?
   282 co  0:02 ps
(D1;Save)->
```

4.5.4.19 Enter Comments

' comment

This command is used to enter a comment. To enter a comment, type the ' interactive test command followed by a comment.

4.6 Disk Parameters File, *DB_diskfmt* Description

The disk parameters file, */mnt/bin/lib/DB_diskfmt*, contains information about disk drives. This information is required to be able to format or test new drives. Each line in the file contains a number of fields separated by one or more spaces. Comments start with a # and continue to the end of the line. Blank lines are also allowed. To add new drives, create a new entry with all the fields defined, and then add the drive to the */ioconfig* file. As of the time of this printing, *DB_diskfmt* contains the following information for all CONVEX machines:

Figure 4–46, Contents of the *DB_diskfmt* File

```
# DB_diskfmt - file of disk parameters
# >>>> WARNING - DO NOT USE 'diskfmt' TO FORMAT! It is no longer compatible
# >>>>           with the CONVEX FORMAT! Instead, format MBUS-attached drives
# >>>>           with 'dev4110' and VME-attached drives with 'dev5130'.
# KEY FOR DRIVE NAMES (unformatted capacity is given in parentheses):
#  Name          Description            Name          Description
# DKD-001       Fujitsu Eagle (452MB) DKD-008,208  NEC 2363 (1080MB)
# DKD-002       CDC 9766 (300MB)      DKD-214      Hitachi DK514-38 (356MB)
# DKD-005,206   NEC 2352 (500MB)
#------- XYLOGICS 450/451 SMD CONTROLLER (MBUS) -------
#  a     b    c   d  e  f    g     h    i j k  l  m   n   o  p
DKD-001 2   842 20 46 45 4800 28160 1 0 1  0  0  smd mfm y
DKD-002 0   823 19 32 31 5040 20160 1 0 1  0  0  smd mfm y
DKD-005 0   760 19 60 59 4832 36288 1 0 1  0  0  smd 2-7 y
DKD-008 1  1024 27 68 67 4816 40960 1 0 1  0  0  smd 2-7 y
#------- INTERPHASE 4201 ESDI CONTROLLER (VME) --------
#  a     b    c   d  e  f    g     h    i j k  l  m   n   o  p
DKD-214 0   903 14 51 50 4736 30240 5 5 1  8  8 esdi 2-7 n
#------- INTERPHASE 4200 SMD CONTROLLER (VME) ---------
#  a     b    c   d  e  f    g     h    i j k  l  m   n   o  p
DKD-206 0   760 19 60 59 4832 36288 5 4 1 12 12  smd 2-7 n
DKD-208 0  1024 27 68 67 4816 40960 5 6 1 16 12  smd 2-7 n
# LEGEND:
# a - drive name                Must be DKD-0XX for Multibus and DKD-2XX for
#                               VMEbus
# b - disk type                 For Xylogics controller
# c - # of cylinders
# d - # of heads
# e - # of physical sectors     Number of actual sectors excluding runt
# f - # of logical sectors      Number of physical sectors (e) minus number of
#                               spares
# g - bits per sector           Number of bits between sector pulses
# h - bytes per track           Total number of unformatted bytes per track
# i - skew                      Sector offset from one head to the next
#                               Must be 1 when using Xylogics 450/451 cntlr
# j - # of relocation tracks    .5% of number of cylinders (c). Raise
#                               fractional part to next higher whole number.
#                               Ignored by dev4110 (Multibus formatter)
# k - interleave                sector separation between consecutive sectors
#                               Currently must be 1 for Xylogics 450/451 cntlr
# l - gap 1 size                Number of halfwords in gap before header
#                               (2 bytes per halfword)
#                               Ignored by dev4110 (Multibus formatter)
# m - gap 2 size                Number of halfwords in gap following header
#                               (2 bytes per halfword)
#                               Ignored by dev4110 (Multibus formatter)
# n - drive interface           Used to determine how to read manufacturer's
#                               defect map.  Currently, smd or esdi
#                               Ignored by dev4110 (Multibus formatter)
# o - data encoding scheme      Way data is encoded on the media. Used to
#                               select patterns for pattern test.
#                               Currently mfm, 2-7 or 1-7
# p - Are spares interleaved    For Xylogics, 'y'. For Interphase, 'n'.
```

4.7 Diagnostic Cylinder Description

The diagnostic cylinder is located on the first full cylinder preceding the sector forwarding area. Cylinder 840 is the diagnostic cylinder on a Fujitsu Eagle drive, and cylinder 820 is the diagnostic cylinder on a CDC SMD drive.

Each track of the cylinder is independent of the other tracks since there is a "table of contents" stored at sector 0 of each track This table describes what is stored on the remainder of that track. The format of this table is shown below:

Figure 4–47, Diagnostic Cylinder Table of Contents Format

```
/* ------------------------------------------------------------------- *
 *              Diagnostic Cylinder Definitions                        *
 * ------------------------------------------------------------------- */
#define NOT_USED        0x00000000        /* position in tbl not used     */
#define NO_HDR          0x00000001        /* this sectors header deleted  */
#define TBL_CONT        0x00000002        /* sector contains tbl of cntnts*/
#define ECC_ERR         0x00000003        /* sector written with bad ecc  */
#define PATTERN         0x00000004        /* sector written with pattern  */

 struct   tbl_cont {
    int   magic_nbr;            /* identifies this as a table of contents */
    int   version;              /* version number of this table */
    int   cyltksc;              /* sector header for this sector*/
    struct {
        int   sec_cont;         /* defines contents of corresponding sector */
        int   pattern;          /* pattern or mask used to verify contents  */
    } sec_tbl[62];
    int   checksum;             /* arithmetic tally of all the above fields */
 };
```

Each track on the cylinder is formatted in such a way that sector 0 is always on the track; sector 0 is written and verified without error when the drive is formatted.

The first field in the table contains the magic number 0x84101500. If this number is not present in the first position of a table, then someone has altered the diagnostic cylinder.

The second field in the table contains the version number. This number is an ordinal number starting with one and increasing as needed. Its purpose is to allow some measure of compatibility of newer diagnostic cylinders with older software. This number is checked if an unknown entry in the sector table (sec_tbl) is encountered. If the version number in the table is greater than the version number of the software, the diagnostic cylinder is assumed to have been formatted with a newer revision of software, and the opcode in the sector table is new to the older software. Therefore, the software is not familiar with the opcode, and the entry in the sec_tbl is ignored.

The next field in the table contains a copy of the sector header (cyltksc) for the table of contents.

The next field in the table contains the sector table (sec_tbl). This table describes what is stored on the rest of the sectors in this track. The sector table consists of 62 entries of 2 int's. Each entry (0 - 61) in this table, corresponds to a sector (0 - 61) on the track. The first position in each entry is the sector contents (sec_cont) field. If the track has more than 62 sectors, the extra

sectors are not used or checked by current software. The following table describes what is currently defined:

Table 4–9, Defined Values for Sector Contents Field

FIELD	DESCRIPTION
NOT_USED	This sector has not been initialized and should not be checked.
NO_HDR	When this sector is read, the controller should return a 0x05 sector not found error.
TBL_CONT	This sector contains a copy of the table of contents.
ECC_ERROR	This sector contains a copy of the table of contents that has been corrupted to give some form of ECC error. The pattern field contains a mask which may be xor'ed with the magic number field to correct the error. If the mask contains 11 or fewer one bits, the controller should report a soft ECC error when this sector is read. If the mask contains more than 11 one bits, the error reported should be a hard ECC error.
PATTERN	This sector has been filled with the data contained in the pattern field. No errors should be encountered when the sector is read.

The last field in the table of contents is a checksum. This is an arithmetic tally of all the bytes in the table of contents. This value is checked before any of the data in the table of contents is used.

4.8 Error Codes

The *dev4110* test reports controller and drive errors in a standard format. However, if additional data is available at the time of the error, it reports that data also. The basic format for the errors is:

```
dev4110 ERROR (Err) ccu/mb/csr/d=c/m/rrr/d Err_desc
```

where:

(*Err*)	Identifies the error code associated with the description.
Err_desc	Describes the type of error that occurred.
c	Gives the CCU slot number for the IOP.
m	Is the Multibus chassis number.
rrr	Gives the control and status register address (CSR), which identifies the board.
d	Specifies the drive number. Each drive begins at 0 with additional drives of the same type being numbered 1, 2, and 3. The number 4 indicates the error occurred while performing a Set Drive Size or NOP command.

If no drive is associated with the error, then the controller and drive information does not appear in the error message.

Additional data may also be reported on a second line (depending on the type of error). For example, the format for data compare errors is:

```
Cyl:dddd Hd:dd
```

The following format appears when errors involve *write, read, read* and *write HDE, read* and *write track headers,* or *seek* commands (depending on the error, the sector number or number of seeks may not appear in the error message):

```
Cyl:dddd Hd:dd Sect:ddd Disp:ddd Exp:hhhh Act:hhhh #Seeks:d
```

where:

Cyl	Specifies the position of heads on the drive.
Hd	Identifies the head selected.
Disp	Identifies number of bytes from the beginning of the sector (data compare errors only).
Exp	Specifies the expected value (data compares errors only).
Act	Specifies the actual value (data compares errors only).
Sect	Indicates the sector where the error occurred.
# Seeks	Identifies the number of seeks executed.

4.9 Error Messages

Both *dev4100* and *dev4110* generate the same error codes and report errors in the same format. For details on common error codes and their descriptions, refer to the section entitled **Error messages** in the *dev4100* test description.

In some circumstances, a certain set of error messages are displayed if the peripheral is not powered on, is write protected, or is cabled incorrectly. The test gives the user several options to proceed with, depending on the circumstances. The following examples illustrate these type of error messages and user options that are presented. The first example illustrates the messages produced when the peripheral is write protected and an operation attempts a write. The second example illustrates the messages produced when the peripheral is not ready (ie., power is not on or a cable is not connected).

> NOTE
>
> In the following examples, all **boldface** entries are user entered information.

Figure 4–49, Drive Not Ready Error Example

```
(D2;Nosave)-> verify_format 0 0 0
             Drive 2: Read of copy #1 of BBT from cyl:759 hd:18 sec:34 failed

dev4110 ERROR (0x16)-ccu/mb/csr/d=7/1/3f0/0 Drive is not ready or faulted
  Cyl: 759 hd:18 Sect: 34
IOPB: cmd stat  drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
     d200 8516 0420 1222 f702 0500 1410 0050 0000 0000 0000 0000
      Drive 2: Read of copy #2 of BBT from cyl:759 hd:18 sec:39 failed

          .
          .
          .

dev4110 ERROR (0x16)-ccu/mb/csr/d=7/1/3f0/0 Drive is not ready or faulted
  Cyl: 759 hd:18 Sect: 49
IOPB: cmd stat  drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
     d200 8516 0420 1231 f702 0500 1410 0050 0000 0000 0000 0000
      Drive 2: Read of copy #5 of BBT from cyl:759 hd:18 sec:54 failed

dev4110 ERROR (0x16)-ccu/mb/csr/d=7/1/3f0/0 Drive is not ready or faulted
  Cyl: 759 hd:18 Sect: 54
IOPB: cmd stat  drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
     d200 8516 0420 1236 f702 0500 1410 0050 0000 0000 0000 0000

dev4110 ERROR (0x4d)-ccu/mb/csr/d=7/1/3f0/0 All copies of BBT are Bad
The internal copy of the BBT is cleared since no good copies were found
Do you want to retry the last operation [yn] -> n
verify_format: pass # 1
dev4110 ERROR (0x4d)-ccu/mb/csr/d=7/1/3f0/0 Drive is not ready or faulted
  Cyl:   0 Hd: 0 Sect:  0 # Seeks:1
  IOPB: cmd stat  drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
       d200 8516 0420 0000 0000 0100 1410 0050 0000 0000 0000 0000

dev4110 ERROR (0x16)-ccu/mb/csr/d=7/1/3f0/0 Drive is not ready or faulted
  Cyl:   0 Hd: 0 Sect:  0 # Seeks:1
  IOPB: cmd stat   drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
Attempt to fix the problem. You can issue UNIX command if it helps.
Then enter one of the following:
        <CR> to retry the operation
        'nofix' if unable to fix the problem
UNIX command, ,CR> for retry, or nofix: RETURN

dev4110 ERROR (0x16)-ccu/mb/csr/d=7/1/3f0/0 Drive is not ready or faulted
  Cyl:   0 Hd: 0 Sect:  0 # Seeks:1
  IOPB: cmd stat   drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
Attempt to fix the problem. You can issue UNIX command if it helps.
Then enter one of the following:
        <CR> to retry the operation
        'nofix' if unable to fix the problem
UNIX command, ,CR> for retry, or nofix: noflx
```

Appendix A
Reporting Problems

A.1 Overview

This appendix introduces the CONVEX Technical Assistance Center (TAC) and the *contact* utility. The *contact* utility is an online system for reporting problems to the TAC. To learn *contact* by using it, enter **contact** at the system prompt and then answer the questions as they appear on the screen. To find out more about using *contact*, read through this appendix. It describes prerequisites and tips for using *contact* and the step-by-step process *contact* takes you through.

A.2 Technical Assistance Center

The CONVEX Technical Assistance Center (TAC) is staffed by technical specialists who can address the diverse questions and problems that arise in a supercomputing environment. If you have a hardware, software, or documentation problem, contact the TAC. This group stands ready to solve such problems.

A.3 The *contact* Utility

The TAC recommends using the *contact* utility to report a hardware, software, or documentation problem. The *contact* utility is an interactive utility that helps the TAC track reports and route them to the the CONVEX personnel most qualified to fix them.

After invoking *contact*, it prompts for information about the problem. When you finish your report, *contact* electronically mails it to the TAC. You are notified within 48 hours that the TAC has received your report.

A.4 Prerequisites

To use *contact* requires

- a UNIX-to-UNIX Communication Protocol (UUCP) connection to the TAC
- the full path name of the program or utility in question
- the version number of the program or utility in question

A.4.1 UUCP Connection

Before using *contact*, check with your system administrator to be sure there is a UUCP connection to the TAC. A UUCP connection allows files to be copied from one UNIX system to another. The *uucp* (UNIX-to-UNIX copy) command relies on either a dial-up or hard-wired UUCP communication line.

A.4.2 Finding the Program Path Name

To determine the full path name of the program or utility in question, use the *which* command. The following screen illustrates using the *which* command to find the full path name of the loader (*ld*) utility:

```
>which ld
/bin/ld
>
```

In this example, the full path name of the loader is */bin/ld*.

For more information on the *which* command, refer to the *which*(1) man page. You can also use the *info* online information system. Enter **info which** at the system prompt. If you use the C shell (*csh*), you can also use the *whence* command to find the program path name. The *whence* command works like *which*, only faster.

A.4.3 Finding the Program Version Number

To determine the version number of the program or utility in question, use the *vers* command. The following screen illustrates using the *vers* command (enter **vers**, then the path name of the program or utility) to find the version number of the loader (*ld*) utility.

```
>vers /bin/ld
/bin/ld: 7.0
>
```

In this example, the loader utility version number is 7.0.

For more information on the *vers* command, refer to the *vers*(1) man page. You can also use the *info* online information system. To do so, enter **info vers** at the system prompt.

A.5 Tips on Using the *contact* Utility

The *contact* utility is interactive and easy to use. This section lists tips to help use it efficiently. In particular, this section tells how to

- use a *.contact* file
- abort a contact session
- resubmit an aborted report
- suspend a contact session
- move from one prompt to another
- use tilde-escape sequences in the *contact* utility

A.5.1 Using a *.contact* File

When invoked, *contact* prompts for information regarding the problem. The first prompt is for your name, title, phone number, and company name. You can, however, create a *.contact* file to skip this first prompt. Follow these steps:

1. Create a *.contact* file in your home directory.
2. Enter your name, job title, phone number, and company name, each on a new line.

When you invoke *contact*, it automatically includes the *.contact* file as input for the first prompt and proceeds to the next prompt.

A.5.2 Aborting the Report

To abort a contact report, either enter the interrupt key (usually CTRL-C) or choose the abort option when prompted by the *contact* utility. Using CTRL-C to abort does not save the contents of the report. Using the abort option saves the contents of the report in a file named *dead.report* in your home directory.

A.5.3 Submitting the *dead.report* File

When aborting a contact session, the *contact* utility saves the report in a file named *dead.report* in your home directory. Using the *contact* command with the *-r* option automatically merges the contents of the *dead.report* file into the new contact session. Enter

contact -r

and *contact* finds the *dead.report* file in your home directory and merges it into the contact report. You can then edit the report. When you end the editing session, *contact* returns to the final prompt, which asks you to review, edit, submit, or abort the report.

A.5.4 Suspending a Report

Sometimes it is necessary to stop in the middle of a contact report and return to the shell (for instance, to suspend the contact session to find the program path name or version number). To suspend the contact session, press CTRL-Z. To return to the contact session, enter **fg**. Using CTRL-Z and the *fg* (foreground) command lets you switch back and forth between the *contact* utility and the shell. You cannot, however, use CTRL-Z and *fg* to switch back and forth if you are using a Bourne shell (*sh*).

A.5.5 Ending a Response

The *contact* utility prompts for information pertinent to your hardware, software, or documentation question. Some prompts require one-line responses; to move to the next prompt, press RETURN. Other prompts require more than a one-line response; to move to the next prompt, press CTRL-D.

A.5.6 Tilde-Escape Sequences

The *contact* utility treats input beginning with a tilde (~) as a special sequence. The character following the tilde is considered a request for a special function. The following tilde sequences are recognized by *contact*:

~e	Start the text editor (defined in your EDITOR environment variable).
~h	Display a list of available tilde-escape sequences.
~p	Print the contact report to the terminal screen.
~r *filename*	Read the contents of *filename* as a response to the current prompt. Some prompts require only a one-line response. This tilde-escape sequence only works for prompts that allow more a than one-line response.
~~	Insert a single tilde as the first character in the line.

A.6 Using the *contact* Utility

The *contact* utility prompts for the following information:

- your name, title, phone number, and corporate name
- the name and version of the product involved
- a one-line summary of the problem
- a detailed description of the problem
- the priority of the problem
- instructions on how to reproduce the problem
- comments about the problem
- comments about the documentation supporting the problem
- files to include in the contact report

The following is a step-by-step discussion of these prompts:

1a. To invoke the *contact* utility, enter **contact** at the system prompt. The system responds with a welcome message and a series of questions regarding your hardware, software, or documentation question. The following screen illustrates the *contact* command and the system response:

```
>contact
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
>
```

1b. If there is a *.contact* file in your home directory, *contact* skips the first prompt. The following screen illustrates the *contact* command and the system response when a *.contact* file is in your home directory:

```
>contact
Welcome to contact version 0.11 ()

Enter the name of the product involved
>
```

2. The *contact* utility prompts for the version number of the product. If you do not know the version number, use CTRL-Z to suspend the session. Use the *which* (or *whence* if using *csh*) and *vers* commands to find the version number of the product. Use the *fg* command to return to the session and enter the version number in the form X.X or X.X.X.X.
3. The *contact* utility prompts for a one-line summary of the problem. This summary is the subject header in any further correspondence regarding the problem. Make this summary as descriptive as possible in one line.
4. The *contact* utility prompts for a detailed description of the problem. Make this description as complete as possible. Include source code and a stack backtrace whenever possible. (Refer to the *adb*(1) or *csd*(1) man page for information on obtaining a stack backtrace.) The more information provided, the quicker the TAC can isolate and solve the problem.
5. The *contact* utility prompts for the priority of the problem. The following screen illustrates this prompt and the priority levels from which to choose; you must enter a priority number.

```
Enter a problem priority, based on the following:
1) Critical    - work cannot proceed until the problem is resolved.
2) Serious     - work can proceed around the problem, with difficulty.
3) Necessary   - problem has to be fixed.
4) Annoying    - problem is bothersome.
5) Enhancement - requested enhancement.
6) Informative - for informational purposes only.
>
```

6. The *contact* utility prompts for an explanation of how to reproduce the problem. Include the command syntax and options you used and anything else you did to make your program run.
7. The *contact* utility prompts for any other pertinent comments. Include any relevant information.
8. The *contact* utility prompts for suggestions regarding the documentation supporting the product. Indicate if the documentation could be revised to address the question.
9. The *contact* utility asks for the names of files necessary to reproduce the problem. The following screen illustrates the *contact* prompt and sample user response:

```
Are there any files that should be included in this report (yes | no)?
>yes
Please enter the names of the files, one to a line (^D to terminate)
>test.f
>~/subroutines/sub.f
>
```

NOTE

Tilde-escape sequences are not recognized in responses to this prompt. Instead, *contact* treats a tilde in this section to mean your home directory. This convention is based on use of the tilde for expanding file names in *csh*.

If the files specified are small text files, they are automatically included in the contact report. If the files are too big to be included in this report, *contact* gives further instructions on how to submit these files.

To specify a directory, combine the directory files into a single file using the *tar* command (refer to the *tar*(1) man page for further information) or enter each file name in the directory on a single line in the contact report.

10. The *contact* utility prompts you to review, edit, submit, or abort the contact report. The following screen illustrates this prompt:

```
Please select one of the following options:
1) Review the problem report.
2) Edit the problem report.
3) Submit the problem report.
4) Abort the problem report.
>
```

Choose the number of the option you want to select. These options let you do the following:

Review — Review the text of your contact report. You are then prompted again to select an option.

Edit — Edit the text of the contact report. If you choose to edit the report, *contact* puts you in your default text editor.

Submit — Send the report to the CONVEX TAC. You are notified within 48 hours that the TAC has received the report. This option exits the *contact* utility and returns you to the shell environment.

Abort — Save the text of your report in a file named *dead.report* in your home directory. This option exits the *contact* utility and returns you to the shell environment.

Index

S

T

U

V

W

CONVEX Multibus Storage Module Drive (SMD)
Disk Formatter (*dev4110*) Diagnostics Manual
Document No. 760-002030-000
First Edition

Reader's Forum

Please use this form to submit comments or questions concerning the clarity and service of this manual. Constructive critical comments are most welcome and help us continue in our efforts to generate quality customer documentation. Please list the page number for questions or comments.

_____

_____

_____

_____

_____

_____

_____

_____

From:

Name_____ Title_____

Company_____ Date_____

Address and Phone No._____

FOR ADDITIONAL INFORMATION OR DOCUMENTATION:

Location	Phone Number
From all locations in continental U.S.	1(800)952-0379
From locations in Alaska & Hawaii	1(214)497-4379
From locations in Canada	1(800)345-2384
From all other locations	Contact nearest CONVEX office

Direct mail orders to:

CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson TX 75083-3851 USA

(Fold Here First)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson TX 75083-3851

(Fold Here Second)

(Tape or Staple)